April 2014

Dangerous Inspection & Versatile Exploration Robot (DIVER): Tracking, Monitoring and Assisting Human Divers in Commercial, Environmental and Military Applications

Christopher Harold Conley
Worcester Polytechnic Institute

Gregory John Hutchinson
Worcester Polytechnic Institute

Jillian Louise Chalke
Worcester Polytechnic Institute

Paul Francis O'Brien
Worcester Polytechnic Institute

Victor Charles Puksta
Worcester Polytechnic Institute

Project Number: MQF-MQP 3120

Dangerous Inspection & Versatile Exploration Robot (DIVER): Tracking, Monitoring and Assisting Human Divers in Commercial, Environmental and Military Applications

A Major Qualifying Project
Submitted to the Faculty of the

WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Bachelor of Science
By

Jillian Chalke

Robotics Engineering

Paul O'Brien

Mechanical Engineering

Christopher Conley

Robotics Engineering

Victor Puksta

Mechanical Engineering

Gregory Hutchinson

Mechanical Engineering

MIRAD Laboratory, August 30, 2013 – May 1, 2014

Approved by:
Professor Mustapha S. Fofana, PhD, Advisor
Director of MIRAD Laboratory, Mechanical Engineering Department

Abstract

The use of professional scuba diving teams is an industry practice for multiple commercial, environmental and military applications. Professional divers can be deployed into dangerous conditions where surface communication is often limited. In these situations, remotely operated vehicles (ROV's) can be used to improve communication and ensure personnel safety and mission success. The objectives of the MQP focus on designing and building a Dangerous Inspection and Versatile Exploration Robot (DIVER) that is capable of assisting, tracking and monitor professional divers in commercial, military and environmental research applications. The DIVER ROV is 22 inches long by 26 inches wide and weighs 17.2 pounds. Using a combination of commercially available and custom made components, the DIVER team created a user friendly and highly mobile ROV platform. The team conducted background research on the state of the art of ROVs, and determined specific applications and features necessary for the DIVER ROV. Upon the conclusion of the background research, the team designed and prototyped hardware and software components for the DIVER ROV. The team used an advanced real time tracking algorithm, OpenTLD, to provide the ROV the ability to track scuba divers during a mission. By integrating a forward facing camera, the DIVER was able to facilitate monitoring by surface operators. Additionally, by providing a means of communication to the surface, the DIVER ROV facilitate rapid assistance in case of emergency. The final DIVER ROV is expandable and deployable while maintaining the ability to increase mission safety and effectiveness through assisting, tracking and monitoring.

Table of Contents

List of Figures

List of Tables

ACKNOWLEDGEMENTS

Our group would like to express our appreciation for our project advisor, Mustapha Fofana whose contributions helped to make this project possible. Additionally, we would like the members of the WPI machine shop, including Matthew DiPinto, and MIRAD Laboratories for their guidance in the logistics of the DIVER ROV project. Thank you all for your insight throughout this project, we are truly grateful for all of your time and effort.

CHAPTER 1: DIVER ROV Project Overview

1. Introduction

Within the diving industry there is a need for improved surface to diver communication and diver monitoring systems. Wireless underwater communication is very expensive and additional monitoring divers raise the risk of the mission success. Environmental conditions can endanger professional scuba diving teams and delay response time from rescue personnel in the event of a medical emergency. Professional divers are often subject to hazardous missions in which there are severe restrictions from rapidly returning to the surface. In the event of a medical emergency, a professional diver may not be able to directly return to the surface due to a phenomenon known as rapid decompression sickness. This causes nitrogen bubbles to form in the blood because the lungs cannot expel the gas quickly enough. Decompression sickness can result in blood clot which, if not treated immediately, can cause death. For these reasons, surface to diver communication needs to be increased.

The objective of this project focuses on designing a Remotely Operated Vehicle (ROV) that can track, assist and monitor divers in commercial, research and military applications. The ROV, which is named Dangerous Inspection & Versatile Exploration Robot (DIVER), in designed to improve safety and communication during underwater diving. The DIVER ROV accompanies the human diver on missions, tracking his/her location through OpenTLD software. In addition, the DIVER ROV is capable of providing live video footage to a surface station. It communicates messages from the surface operator to the underwater diver in the form of messages on a LED array. Furthermore, the monitoring capability that the DIVER ROV provides decreases the number of divers needed in an operation. Given the dangerous nature of diving, less divers means decreased risk as well as reduced operational costs. Surface operators will be able to better understand each mission by actively participating via the robot.

The DIVER team accomplished the objective of the project through extensive design process as outlined in this report. The team conducted in-depth research of industry leading ROV technology and the operational conditions of ROVs. The DIVER goal and purpose was established and design specifications were selected. The DIVER specifications included being able to operate in depths from 0 to 50ft and water temperatures between 50 and 70F. The team designed and fabricated a frame based on the aerodynamic design of an airfoil and designed an intensive electronic control system that utilizes onboard microcontrollers and tethered communications and power systems. Commercial components such as sensors and waterproof dry boxes were combined with fabricated components such as frame pieces to develop the DIVER ROV. The completed DIVER ROV measured 22 inches long by 24 inches wide by 9 inches high, and weighed 20 pounds. The team executed a three stage testing process consisting of individual component testing, full system dry testing and full system aquatic testing. Both tele-operational and autonomous capabilities were tested in addition to mobility and camera image quality. Observations were recorded in order to create a final summarization and future recommendations in the report.

The remaining part of the report is categorized as follows. In Chapter 2, a class of ROVs and their applications are presented. Chapter 2 specifically provides background research on ROVs on the cutting edge of industry, including materials, communication and propulsion, as well as the environment that they operate in. Chapter 3 thoroughly outlines the design, construction and programming of the DIVER ROV. Finally, Chapter 4 discusses the results of the project and recommendations for future progress and direction.

CHAPTER 2: Remotely Operated Vehicles and Operational Conditions

2. Introduction

The DIVER ROV system was designed to assist and monitor professional and recreational scuba diving personnel, providing improved surfaced monitoring and communication, increasing overall safety the system provides divers with assistance capabilities and functionality to alert underwater personnel to environmental hazards. To ensure the maximum functionality of the DIVER ROV, it was necessary to understand the diving and ROV industries. Research was conducted on multiple commercial and government applications, focusing on areas in need of improved monitoring and communication processes.

2.1 Applications of Remotely Operated Vehicles

When determining the application for this project, it is crucial to consider existing ROV applications and find industry limitations which can be expanded by new ROV technologies. As described previously, the aim of this project is to design a partially autonomous underwater robot which will follow and assist a diver, as well as monitor the diver's safety. Implementing such a robot will improve underwater communication between the diver and the surface, as well as reducing the cost of typical underwater jobs performed by skilled divers. Because divers typically function in pairs, replacing one of the divers with a robot that can act as an assistant will greatly cut down on the overall cost. Furthermore, one of the main risk factors that come into play for underwater diving tasks is poor surface to diver communication. Because wireless transmission through water is challenging, it is not always feasible for voice communication between the divers and the surface crew. By implementing a system that can alert the diver with optical signals transmitted from the surface through the robot, these accompanying risks can be greatly reduced. In order to refine and develop the project's goal and scope, it is prudent to examine not only existing ROV technological applications, but also to research typical diver tasks—and the risks

that go along with these tasks—in order to best tailor the project to existing industry needs, and best design an ROV that can improve these areas. Additionally, a main source of engineering innovation in the present day stems from studying past technologies to gain inspiration and to improve upon such technologies. This makes researching industry standards and current technologies a highly relevant part of the engineering process.

2.1.1 Technological Applications of Remotely Operated Vehicles

Present day ROVs can be divided into two main classes—observation class and working class. Observation class ROVs are designed mainly for underwater visualization, while working class ROVs are used for applications which require manipulation. Because of their more intensive applications, working class robots are typically much larger and more expensive, and have much greater power requirements. The DIVER ROV MQP is much more relevant on observation class robotic applications. Figure 1 shows the evolution of ROVs descending from underwater vehicles and eventually splitting into these two classes (Christ [13]).

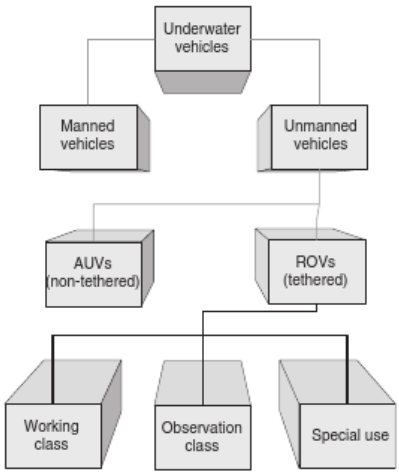


Figure 1: Evolution of ROV Technology

Observation class ROVs are intended to be an extension of the user's senses. In other words, observation class ROVs provide a way for humans on the surface to view and explore underwater areas without being there in person. Observation class ROVs have a wide variety of

applications that include examining and inspecting underwater man-made structures, surveying the underwater landscape for scientific purposes, as well as exploration of shipwrecks, areas of archeological interest, and resource rich locations. Perhaps the only drawback of observation class ROVs is their limited sensor and manipulation capabilities. Because observation-class ROVs are designed first and foremost for underwater visualization, they are typically compact for maneuverability and ease of use. There are many applications in which manipulation is not an issue where observation class ROVs fulfill the mission requirement (Christ [13]). Sensor packages and manipulator proficiency is often an afterthought. Figure 2 shows a typical example of an observation class ROV.

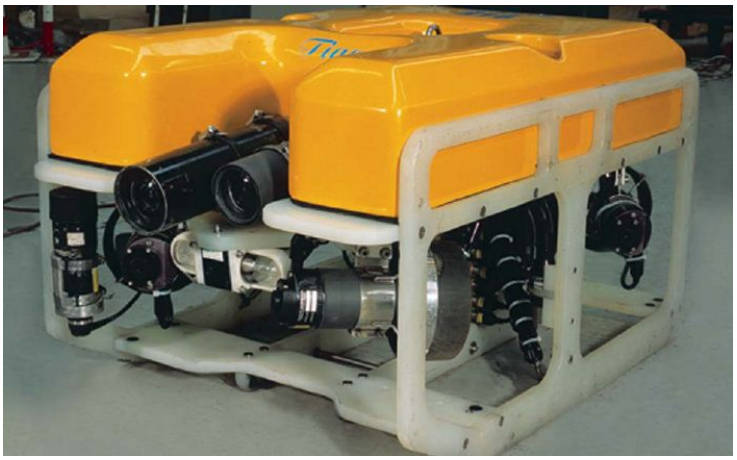


Figure 2: An Observation Class ROV with Lights and Camera

One area of application for observation class ROVs is in Maritime Security. The large scale of military and commercial maritime operations and the growing threat of terrorism have called for increased maritime security measures in recent years. The Department of Homeland Security and the Department of Defense have developed layered operations to protect critical infrastructures, utilizing a variety of tools varying from acoustic devices, sensor deployment, divers and ROVs (Molchan [44]). ROVs provide government and commercial personnel the ability to inspect and search ship hulls, pier pilings, and harbor bottoms both inside U.S. waters and

abroad. Often used side by side with divers, ROVs, especially micro-ROVs, provide major advantages over their human counterparts. First, in dangerous situations, deployment of an ROV accomplishes the mission without putting the diver at risk. This includes scenarios in which water quality is a potential threat to divers. Second, if the ROV is small enough, it has the potential to inspect areas that are unreachable by a diver, such as in between pier pilings (Molchan [44]). In many cases, ROV operations can be set up and deployed faster than divers in SCUBA operations (Molchan [44]).

Observation-class ROVs are also very common in the energy industry, particularly for oil and gas. Prior to oil platform construction, ROVs are typically used to survey locations in order to ensure the location is topographically suitable. Additionally, once a structure is in place, ROVs can inspect underwater structures and ensure they are sound, monitor corrosion, and scan for structural impurities such as pipeline cracks. An example of structural inspection being conducted can be seen in Figure 3, in which a partially autonomous ROV is working on an oil pipeline in the offshore petroleum industry ("Swimming" [66]). This type of inspection can be performed visually by the operator using a camera on the ROV, or the ROV itself can be outfitted with sensors packages making it partially autonomous (Christ [13]).

Figure 3: Partially Autonomous ROV Working on Pipeline

2.1.2 Diving Applications

Examining typical specialized diver tasks enables the project to be tailored to more specific applications. Diving can often be dangerous and expensive. Eliminating just one diver from a typical underwater job can reduce the cost by thousands of dollars. Furthermore, reducing the number of diver's decreases liability and the likelihood of a potential lawsuit should one of the divers be injured on the job. Skilled divers are generally required to have certifications and required number of hours experience depending on the application. For example, saturation divers, divers which perform subsea tasks such as welding, cutting, inspection and rigging, must have at least one year of experience and take a multi-week course to achieve certification. Because many of these applications create heat in proximity to flammable oxygen tanks or lines, these divers are often exposed to dangerous conditions and may require constant health and safety monitoring by a trains life support technician diver who must also have over 2000 hours of training and experience. Additionally, difficulties with wireless transmission through water mean that communicating with the surface often requires divers to surface, extending the time of the job and thus the cost. Because of the level and specialization required of professional divers, their time on the job is very expensive. Saturation divers can make nearly $250,000 dollars per year, making them a costly addition to an underwater job ("Careers" [9]).

Some of the most common professional diving applications include scientific surveying, construction, filming, welding, and cutting ("Careers" [9]). Because diving can be dangerous and expensive, exploring how robots can replace divers in the future is cost-effective and extremely beneficial to the subsea industry. Some diving applications such as scientific surveying and filming are already performed by observation class ROVs like the one in Figure 4 ("Underwater ROV" [71]).

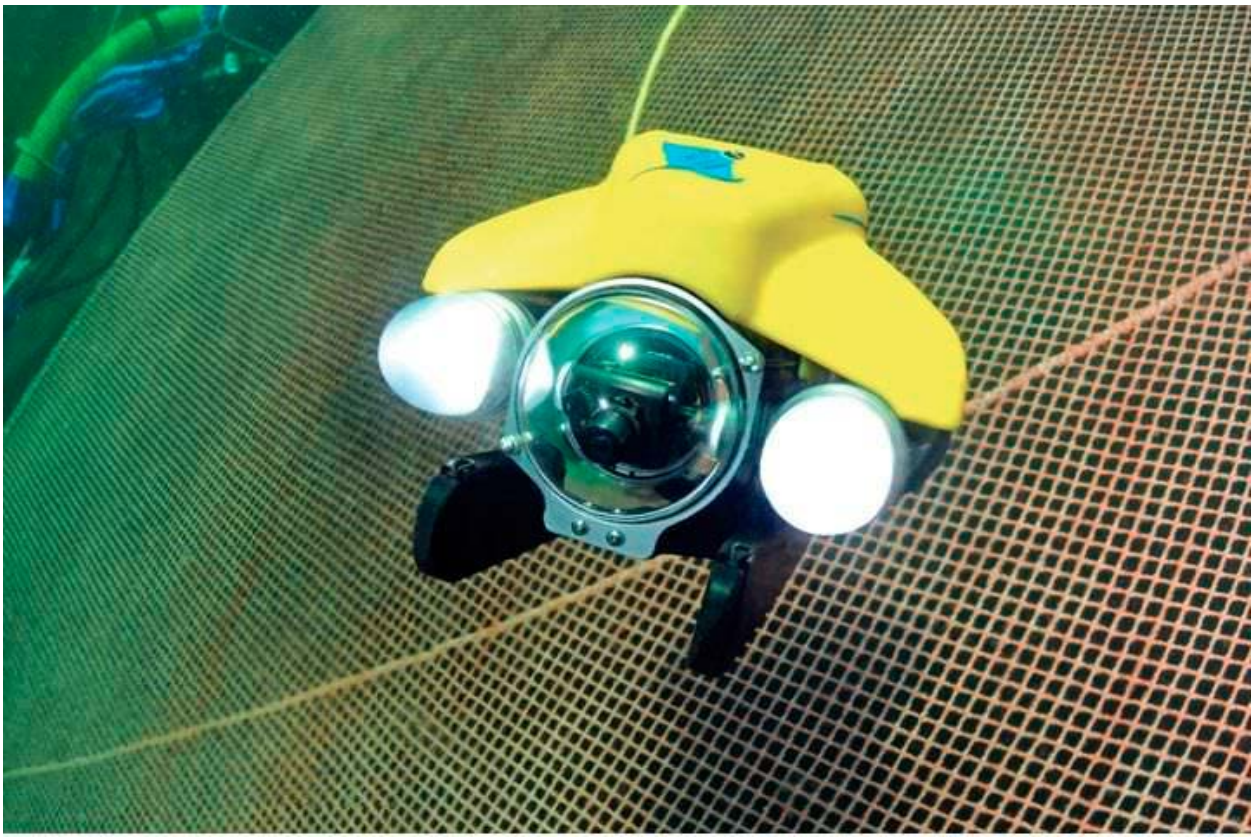


Figure 4: Observation Class ROV

However, skilled divers may not be able to be entirely replaced at this point in time. This being said, replacing at least one diver with an ROV that can carry tools, assist, and monitor the safety of divers, as well as improving surface to diver communication, can greatly improve safety and reduce expenses.

2.2 Aquatic Conditions

Many factors can affect the performance reliability of an underwater robotic system. Conditions including pressure, water depth, salinity and temperature all may affect material choices and structural design characteristics. This section addresses various conditions effecting these decisions.

2.2.1 Pressure and Depth

The water depth to pressure ratio is important to understand for design constraints and material selections for a robotic system. The relationship of water depth to pressure is a linear relationship where pressure increases as depth increases ("Pressure" [56]). The atmospheric pressure at sea-level is 14.7 pounds per square inch absolute (psia). To keep things simple, this has been given unit of 1 atmosphere or ATM. Pressure increases at a higher rate in water than air due to the higher density of water. Specifically, water pressure increases 1 ATM or 14.7 psia every 33 feet (10.06 meters) below the surface of the water ("Pressure" [56]). Table 1 presents pressures at various depths.

Table 1: Water Pressure Depth Chart

Depth (ft.)	Pressure (psia)	Pressure (atm)
0 (Sea Level)	14.7	1
33	29.4	2
66	44.1	3
99	58.8	4
132	73.5	5
165	88.2	6
198	102.9	7
231	117.6	8
264	132.3	9
297	147	10

The operational depth recommendations and restrictions are directly dependent on the pressure that the system will experience for that given depth range. Once a depth range is determine, all materials selected must be able to withstand the pressure at the give operating depth.

2.2.2 Buoyancy Forces and Control

Buoyancy or Buoyant force is a natural phenomenon that occurs according to Pascal's Principle. This principle states that the pressure is a function of the static fluid pressure which is dependent on the vertical location at which the pressure occurs. In the example, a jug of a given height is filled with water and a constant force is applied to the stopper at the throat of the jug. These initial conditions allow for the calculation of the surface pressure at the throat where $\text{Pressure}_{\text{Initial}} = \frac{\text{Force}}{\text{Area}_{\text{CrossSection}}}$. To determine the pressure at the bottom of the jug, the following relationship must be applied, where $\text{Pressure}_{\text{State2}} = \text{Pressure}_{\text{Initial}} + \rho gh$ where ρ is the density of the fluid, g is the gravitational constant and h is the height of the jug. This equation will yield a different pressure at state two than at the original state due to the static fluid pressure that is now accounted for (Nave [48]).

Using the jug example, it is possible to explain the effects that buoyancy will have on an object floating in water. Take the example of a ball of constant material and perfectly spherical. When this pass is placed into a body of water, the top of the ball will experience a different pressure that the bottom of the ball. The vertical location of the top and bottom of the ball relative to the surface will cause unequal application of pressure to the surface of the ball, creating a buoyancy force in the positive y direction. If the buoyant force in the positive y direction is able to overcome the weight force in the negative y direction, the ball will float upward, demonstrating the effects of buoyancy (Nave [47]).

The specific systems of many industry robotics are highly protected because of intellectual property and government regulations. Because of this, many details regarding the buoyancy

systems are closely guarded by the designers and manufacturers. General system overviews of these systems can be found like the one shown below.

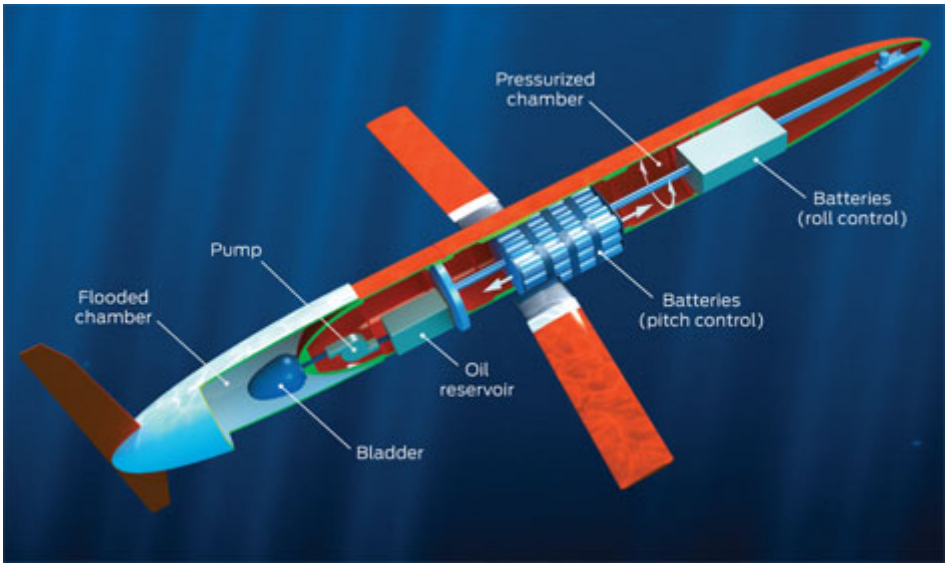


Figure 5: Internal Schematic of the Bluefin Robotics Spray Glider

Figure 5 shows the buoyancy control system for the Spray Glider, an observational class ROV developed by Bluefin Robotics (Taylor [67]). The internal cross section shows the basic internal components of the ROV. The air bladder creates displacement of the water in the flooded chamber, causing the ROV to become more buoyant. Adjusting the amount of water displaced allows the ROV to easily dive or surface (Taylor [67]).

2.2.3 Aquatic Environment Requirements

The aquatic environment that ROVs operate in also dictates the use of certain properties of materials. Water, especially saltwater, easily corrodes certain materials, therefore requiring metal finishing techniques. Polymers and composites have better corrosion resistance, but may not satisfy other requirements such as strength and cost respectfully. Seals and insulation materials must be waterproof to protect the ROV electronics. Additionally, components must be strong enough to withstand the pressure increases involved with diving.

2.3 Material Selections

A wide variety of materials are used for Remotely Operated Vehicles (ROV). Plastics, composites, and metals are used in conjunction with each other to form modern ROVs. The direct application of a ROV directly dictates what materials components need to be. Durability, strength, weight, cost, and corrosion resistance are all factors that need to be evaluated in selecting particular materials for components. Common metals used for ROVs include steel and aluminum. Steel is relatively inexpensive and is very strong. While it is very durable, it is heavy. Aluminum is much lighter than steel, yet less strong and more expensive. Both metals require surface finishing in order to prevent corrosion. Glass Fiber Reinforced Polymers (GFRPs), often referred to as just Fiberglass, are composites used in various applications. Very lightweight materials, GFRPs are composed of glass fibers set in a polymer matrix. The result is an extremely strong material relative to its weight.

The polymer used in Fiberglass is usually a thermosetting epoxy, although polyester is sometimes used. Plastics are very lightweight and inexpensive. Relative to most metals, plastics are easier to manufacture and they can be engineered to possess specific qualities. Unfortunately, polymers alone lack tensile strength and quickly deform under stress. A simple way to increase the tensile strength of a polymer is to reinforce it with a fiber. In the case of Fiberglass Reinforce Polymer, the fiber is glass. Glass fiber has an extremely high tensile strength but is very brittle alone. By setting glass fibers in a polymer to form a composite, the best properties of the two individual materials are combined into one. The resulting composite is still extremely lightweight, but has a high tensile strength.

2.4 Propulsion

The propulsion system of an underwater remotely operated vehicle (ROV) is one of the most important components when designing an ROV and planning the integration of the necessary components. The ROV Manual a User Guide to Observation-Class Remotely Operated Vehicles shows that “the type of thrusters, their configuration, and the power source usually take priority over many other components (Christ [13]). The propulsion system allows for the ROV’s maneuverability underwater. That is, various placements of thrusters on the ROV will allow for increased or decreased maneuver capabilities. Some of the common orientations of thrusters include a three, four, and five thruster system, as seen in Figure 6 (Christ [13). The centered start looking figures represent vertical thrusters while the T looking dual shaded figures represent horizontal thrusters.

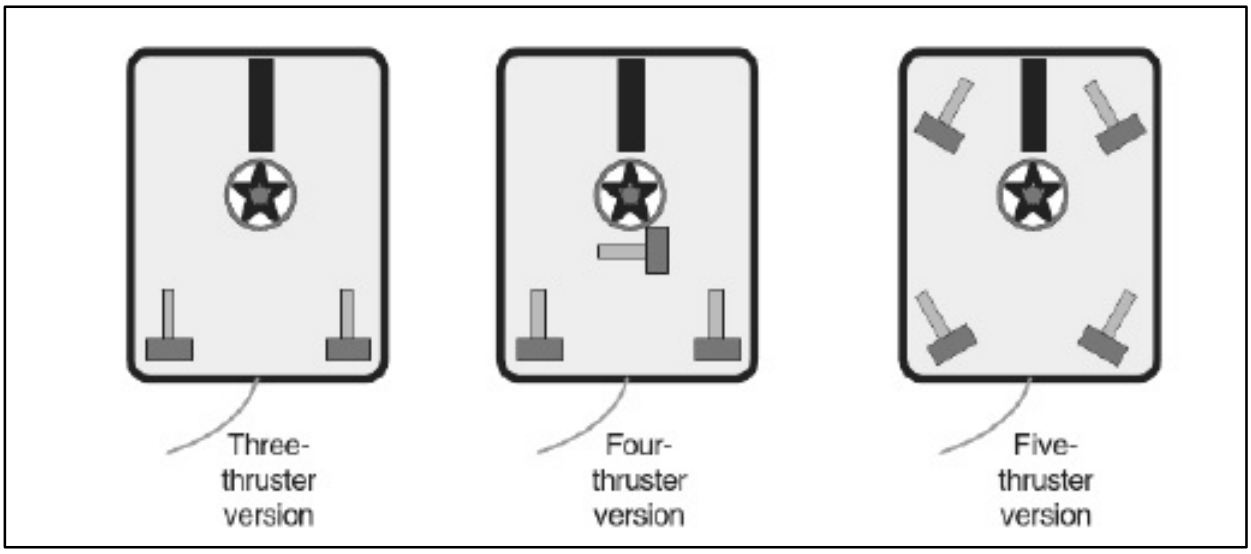


Figure 6: Thruster Orientation

An imbalance of the system will occur if more than one thruster is operating on the same plane of motion. To prevent this, thrusters on the same plane of motion must counter-rotate to prevent the system from creating a turning moment. When building an underwater ROV, three classes of propulsion (thruster) systems are used: electrical, hydraulic, and ducted jet. The

propulsion system for an ROV is chosen based on the specific mission of the underwater vehicle. For example if the ROV will be working in an environment in which loose debris that could get sucked into a thruster, the designer would use a ducted jet system to prevent the debris from affecting the propulsion system. The system selected for an ROV must provide sufficient thrust to compensate for the other components of the ROV, such as weight, drag, and buoyancy.

A common underwater thruster nozzle is the kort nozzle. Also known as a duct nozzle, a kort nozzle utilizes a circular shroud that is hydro-dynamically designed. The reason the kort nozzle is popular is it helps to reduce the number of "vortices generated as the propeller turns at high speeds." It has also been shown to "reduce the incidence of foreign object ingestion into the thruster propeller". The kort nozzle makes for a more efficient thruster by "reducing the tendency of rotating propellers' swirling discharge, which tends to lower propeller efficiency and cause unwanted thruster torque acting upon the entire vehicle" (Christ [13).

2.5 Communication

This section outlines the advantages and disadvantages of both tethering and wireless forms of data transmission for potential use in robotic applications. Additionally, the section covers the various forms of communication used in industry and the selection of communication methods based on mission criteria.

2.5.1 Wireless

Although wireless communication is commonplace for terrestrial applications, aquatic environments present unique and difficult engineering challenges to overcome. There are two primary implementations of wireless technology for underwater use: electro-magnetic waves and acoustic waves. The following sub-sections outline the science behind as well as the difficulties of implementing a wireless form of communication under water.

2.5.1.1 Electro-Magnetic Waves

The use of electro-magnetic communication is very common form of data transmission because of the faster velocity and higher operating frequency of these waves. One major drawback for underwater communication is the behavioral difference electro-magnetic waves experience in salt water and freshwater. In freshwater, electro-magnetic waves have been used to conduct ground penetrating research in freshwater lakes however the size of the antenna on the transmitter determined the effectiveness of the system. As a medium for transmission, freshwater allows for frequency independent transmission and good wave propagation but the size of the transmission antenna makes the system impractical. For the example, an antenna of multiple meters long was considered to be idea for the application. As a salt water application, there is a higher degree of difficulty when transmitting through this medium due to the increase conductivity. This increase allows for very low wave propagation due to the concentration of dissolved solid in the water (e.g. salt). This results in a highly attenuated wave even over relatively short distances. As a result,

industries are currently researching new technologies and methods for improving wireless communications through water.

2.5.1.2 Acoustic Waves

Acoustic wave communication, more commonly known as SONAR, has been the method for underwater communication Acoustic wave communication was first developed for military use during World War I as a means of detecting enemy submarines. The long wavelength acoustic waves makes them ideal for object detection because the discernable Today the acoustic wave communication system has various applications in object detection for commercial and military uses. The disadvantages of data transmission via acoustic waves are frequency-dependent propagation loss, severe multipath, and low speed of sound propagation. The signal is considered to be weak over distance, approximately 1 KHz over 1km which causes it to be subject to a phenomenon known as multipath. This scenario is present is both deep and shallow water conditions and causes signal echoes which result in interference. The wave will reflect off of the surface and the seafloor during shallow water communication and will be subject to ray distortion when in deep water conditions. Additionally, due to the relatively slow speed of sound underwater, the signal efficiency is drastically reduced and is also subject to extreme Doppler distortion. Additionally, it is important to note that background noise found underwater can further corrupt acoustic communication by adding noise to the data that may not be able to completely filter out. While future research is being done to improve the effectiveness of underwater acoustic communications, current technologies are not cost effective or readily available for public use.

2.5.2 Tethered

Tethered communication is considered to be a more robust alternative to wireless communication for robotics applications. Tethering allows for a direct line of communication between the robotic system and the control system. This allows for higher speed of data

transmission between both subsystems as well as little to no data corruption associated with external interference. Additionally, onboard power supplies can be eliminated or reduced because power can be supplied via tether alongside data. This allows designers to reduce weight and cost of the robotic system at a sacrifice of mobility. One of the major downfalls to tethering is the physical issues associated with cable management, especially when in the presence of obstacle navigation.

Additional personnel or equipment may need to be provided to assist in tether management to ensure mission success. For aquatic systems, surface spooling devices are often implemented to reduce underwater cabling. The design team must also be aware of any rotational movements that may cause tethering to tangle within the robotic system or obstacles that could cause damage to the tether, preventing the completion of a mission. In summary, tethering reduced communication issues at a sacrifice to overall mobility.

2.5.3 Communication Systems in Industry

Industry underwater ROV's use both tethering and wireless communication systems, based on the specific mission constraints. Oceanography robots designed to collect data over long periods of time feature wireless communication antennas to transmit data from locations around the world. Figure 7 shows the communication method for Bluefin robotics autonomous underwater vehicles ("Antenna" [2]).

The underwater robot operates autonomously throughout the course of its underwater mission, returning to the surface periodically to transmit data wirelessly. This style of mission allows for indirect control of the robot, reducing the need to communicate between submersions.

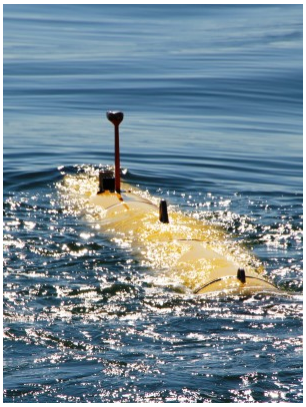


Figure 7: Bluefin Robotics Wireless Transmitter for AUV

Missions that require operator control feature a tethering system. This type of communication is used for inspections where real time user input is required. In underwater inspection, surface operators require feedback from the robot in the form of video feed and sensor signals. Figure 8 shows the remotely operated vehicles featuring a tethering system.

Figure 8: Versatrax 300 a) and Teledyne Bethos Minirover b)

The Verastrax 300 in Figure 8a) is used for internal pipeline inspection while the Teledyne Benthos Minirover in Figure 8b) is used for underwater inspection ("Versatrax" [73], "MiniRover" [43]). Both systems feature tethering systems and tele-operational control. ROV's featuring tethering systems allow for constant user feedback and tele-operational control but feature a limited range. For this reason, the use of wireless and tethering communication systems is specific to the goals and parameters of the mission limitations.

2.6 Autonomy

The following is a brief discussion regarding the benefits and drawbacks of the three levels of autonomy in relation to vehicle control. That is, a tele-operated platform, a semi-autonomous platform, and a fully autonomous platform.

2.6.1 Tele-operated Control

A tele-operated design rests the entire responsibility of control over the vehicle to a human operator. The operator is therefore required to have access to each individual actuated joint, motor, or other system and some means of altering those systems' parameters. In simple systems this eliminates the necessity for complex electronics and software and places control completely in the hands of the operator. However as the complexity of the system increases, so does the difficulty in efficiently and safely operating the vehicle. An apt example to demonstrate the relationship of complexity to relative difficulty controlling a tele-operated vehicle would be that of a toy remote control car to a typical industry standard marine ROV. With the remote control car, through the radio receiver the operator may increase or decrease the throttle, adjust the steering angle, or power the system on and off. The operator gains feedback through his or her eyes as to the performance of the vehicle, thus closing the control loop and the operator is then able to make adjustments accordingly. This is well within the capability of any average, non-skilled operator to do. In the case of a much more complicated system such as a marine ROV, there exist many more functions for the operator to account for than the three used in controlling the car, controlling those functions requires more complicated systems due to the operational environment. Disregarding any specialized tools or manipulators, at minimum, a marine ROV generally exhibits the following mobile characteristics:

- A means to statically or dynamically change depth (It should be noted that it is possible to changed depth by adding an additional degree of freedom regarding attitude control)

- Attitude control in at least 2DOF
- Dual propulsion in at least one axis in relative to the chassis tangential to the vehicle's center of rotation
- A way of providing feedback to the operator

It should be clear from this brief listing that in order to operate a marine ROV, the level of technical knowhow and skill of the operator must be increased proportionally. In examining attitude control alone, this increase in difficulty becomes particularly obvious. Attitude control may be handled by operating propulsion units at differing levels to induce rotation, varying the units orientation in relation to the chassis, utilizing control surfaces (i.e. rudders, hydrofoils, etc.), or any combination of the three. Wherein the remote control car only has one means of changing orientation, many marine ROVS will have several. Therefore a completely tele-operated control system may not be suitable for complex vehicles.

2.6.2 Semi-Autonomous Control

In contrast to a completely tele-operated control scheme, a semi-autonomous control scheme reduces the level of complexity for the operator. A semi-autonomous design is capable of limited computation and part of the control of the vehicle is accomplished through software. The amount of control the software is responsible for may vary significantly depending on the complexity of the vehicle. Examples of the nature of this control are as follows:

- Velocity. Rather than an operator directly controlling voltage and current to the propulsion units, the software may use a PID algorithm to maintain a certain velocity.
- Attitude. The software may use an inertial measurement unit (IMU) along with the vehicle's pressure sensor to automatically maintain a particular attitude within the water column.
- Depth. The software may utilize pressure sensors to determine depth, and adjust the vehicles buoyancy to maintain or change its position in the water column.

In this scenario, the operator takes on the role of a navigator rather than a pilot. Rather than personally adjusting the vehicle's controls, the operator will specify a set of parameters that the control software then enacts. Such a system significantly reduces the amount of training and skill necessary on the part of the operator. The tradeoff in this case is that software must be developed to handle these functions, and sensors installed to provide feedback to the software thus closing the control loop.

2.6.3 Autonomous Control

On the far end of the spectrum from a completely tele-operated vehicle is a completely autonomous control scheme. In this scenario, all of the vehicle's control functions are delegated to software. This requires no operator input during operation whatsoever. The tradeoff is that the vehicle must now be equipped with a suite of sensors capable of monitoring the environment and its own systems. Additionally, the vehicle must be capable of handling complex computations, and fitted with a robust software suite. The advantages to such a control scheme are that there is no chance of operator error, and the vehicle (assuming that it has been designed correctly) can be used by untrained personnel. The disadvantage is that creating the software necessary to handle a dynamic environment as well as carry out a specific task sufficiently is monumental challenge. Also, the expenses involved in equipping a vehicle with the necessary computational power and sensors are significant enough to be a major obstacle for development.

In the case of DIVER ROV, the team felt an objective method of determining the control scheme would be appropriate. Based upon the team's initial research, four main categories were identified as essential to the decision making process. These were expenses (cost in USD), cost in development time, the required training/skill of the vehicle operator, and design difficulty. Using arbitrarily designated values on a scale of one to ten, with one representing the least and ten representing the most, the team created a feasibility matrix (refer to Table 2: Feasibility Matrix).

Table 2: Feasibility Matrix

	Tele-operated	Semi-Autonomous	Autonomous
Cost ($)	2	5	10
Cost of Development Time	2	5	10
Operator Requirements.	10	3	1
Design Difficulty	2	5	10
Totals:	16	18	31

Note that tele-operated and semi-autonomous score within a close margin to each other, while autonomous is close to twice the magnitude of the others. Given this matrix and the design goals of the project, a decision was made to seek to use a semi-autonomous control schema although it scored higher than tele-operation. Had this matrix yielded a more significant margin between the two, the team may have chosen to use tele-operation. However, as one of the key elements of the DIVER ROV design paradigm is its deployability, the benefits of a semi-autonomous design outweigh the disadvantages due to the operator requirement for teleportation regardless of the score. In light of this, the decision to implement a semi-autonomous control schema was made.

2.7 Computer Vision: Object Detection

The problem of accurately extracting or detecting an object in a video stream or digital image has been approached in various fashions for decades. Its prevalence is due in part to the numerous applications for detection across many fields. Four examples of groups using detection algorithms include but are not limited to Law Enforcement, Commercial Entities, Industrial/Manufacturing and Military. Each of these four groups use detection algorithms for a number of purposes. The following is a brief outline of some of the ways they implement the technology.

Law enforcement agencies have begun to make use of facial recognition software to identify suspects in investigations. Funding to develop the technology for further use is provided by the Department of Homeland Security. In addition, vision systems are being implemented for traffic management and enforcement (Iteris [26]). Figure 9 depicts vehicles being monitored with object detection software at a traffic stop.

Figure 9: Detecting license plates for traffic management

Facebook makes use of a facial recognition algorithm codenamed "Deepface" to recommend "tags" for people in photos uploaded to the site. The software purportedly has an accuracy rating of 97.25% which is close to human levels of comprehension (Extreme Tech [20]). Although accurate and convenient to users the storage of biometric data by a commercial entity has created significant controversy and caused a number of countries to attempt to pass legislation regulating how the data is used (Forbes [24]). Other potential uses for object detection in the commercial sector include the myriad of "smart" devices being developed. Regarding industrial applications, usage includes quality control, component sorting, and robot control (Chen et al. [11]).

The military makes use of detection algorithms in a number of ways. These systems are used in unmanned drones (Stanford Univeristy [65]), guided missile systems (“Federation” [21]), and force monitoring (Defense [18]). The paradigm shift towards complete battlefield awareness by technological superiority drives advances in detection algorithms and applications.

2.7.1 Algorithms for Object Detection

The following is a brief discussion of three common algorithms utilized in industry for object detection via computer vision, followed by an in depth dissertation on OpenTLD (our chosen method). A full discourse of every algorithm and its implementation is beyond the scope of this paper. However, each has numerous publications associated with it. Please refer to the listed reference material regarding these algorithms and their implementation.

2.7.1.1 Blob Tracking

Blob tracking (blob detection), is a catchall phrase for a number of algorithms which segment an image into regions of interest based upon properties such as color or brightness (OpenCV [50]). Blob detectors can highly useful and efficient in controlled, low noise environments or with highly specific conditions. Examples include quality control in manufacturing plants where malformed parts can be easily segmented from the sorting environment, or recognized, or an application where a system is used to identify a particular color object. Figure 10 b) below shows blob tracking of the chair in Figure 10 a) ("Programming" [52]).

Figure 10: Detecting a Red Object using Blob Detection

In many cases, blob detection alone is not sufficient for an application and is therefore used as a precursor to segment an image for edge detection or another form of algorithm. For our application, blob detection alone is too susceptible to noise from dynamic environments to be an appropriate detection method and is therefore unsuitable.

2.7.1.2 SURF (Speeded Up Robust Features)

First presented by Herbert Bay in 2006, this feature detector borrows fundamentally from the SIFT (Scale-invariant Feature Transform) but has the advantage of being several times faster. SURF also can be used for the purposes of 3D object reconstruction. At its core, SURF uses 2D Haar wavelet responses and makes efficient use of summed area tables also known as integral images (Bay et al. [4]). While having the advantage of running the fastest of the SIFT family of algorithms, SURF also performs the worst in dealing with changes in blur, illumination, and affine changes (Wu et al. [79]) making it unsuitable for our application. SURF detection can be seen in Figure 11.

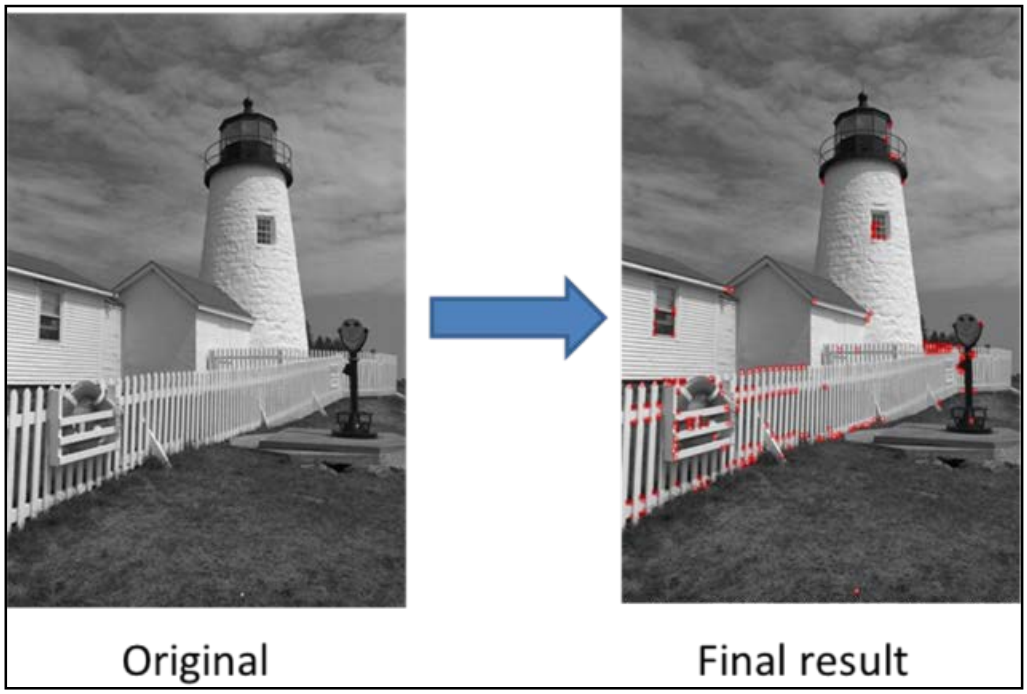


Figure 11: SURF detector

2.7.1.3 Haar Cascade

A Haar Cascade is actually a set of simple classifiers used to detect Haar- like features (OpenCV [50]). Haar like features were first described 1998 as an alternative to working solely with the intensities of each and every pixel of an image (Papageorgiou et al. [52]). Instead, haar-like features are the result of the sums of the intensities of rectangular regions of an image and the differences between them. This concept was implemented into the contemporary algorithm used today (Viola et al. [74]). However, in order to increase accuracy a large number of Haar-like features must be identified within an image. Therefore, the classifier is run several times (hence the term cascade). In order to use a Haar Cascade, a set of training images must be used to develop a model of the features to be tracked. Each of these images must be of the same size and general orientation. Generally, the larger the training set and similarity of the images, the more accurate the detector is. While this method of object detection is highly accurate, it is limited in its scope and susceptible to changes in orientation of the detected object. For example, complete face detection is only possible if a separate model used for front views, isometric views and profile views. If only one of these models is applied, the detector will fail in the other two scenarios.

Trying to generalize the training set with multiple facial views only serves to increase the odds of false detections, as the training set is used to determine common Haar-like features. Although highly accurate given a good set of training images, due to the very specific application of a Haar Cascade it is unsuitable for our application. We would require that classifiers be trained for all of our usage scenarios and then chosen on the fly, which would be impractical as well as computationally expensive.

2.7.2 OpenTLD: Our ideal detector

OpenTLD, also known as the Predator algorithm, was developed by Dr. Zdenek Kalal as his PhD thesis. OpenTLD approaches the problem of visual object tracking by decomposing the process into three sub-tasks. These tasks are tracking, learning and detection (Kalal [30]). What sets this algorithm apart from the previously discussed methods is that this entire process is capable of happening in real time. That is, no prior model must be loaded. The algorithm will generate and adapt a model on the fly based upon a simple user input: a bounding box drawn upon an image. This makes the algorithm highly versatile, adaptive, robust, and easy to implement (Kalal [30]). Figure 12 depicts a breakdown of the three parts of OpenTLD as well as an overview of how they work.

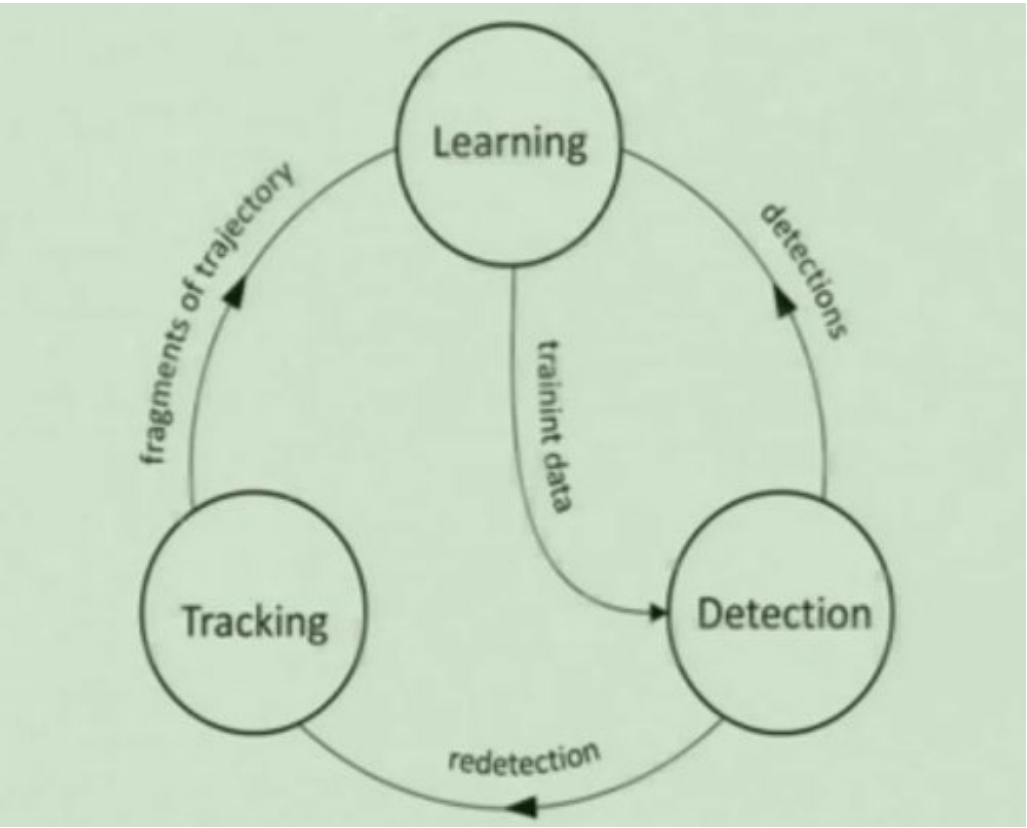


Figure 12: Diagram of the OpenTLD process

2.7.2.1 Tracking

Tracking primarily consists of a Median Flow Tracker augmented with failure detection (Kalal [30]). The tracker is initialized with a bounding box provided by the user. From that point forward, an estimation of the displacement of a number of points within the frame is conducted with the pyramidal implementation of the Lucas-Kanade tracker. A further calculation of an estimation of their reliability is used with Forward-Backward error (threshold cut-off of 50%) to determine the accuracy of the tracked points. As Dr. Kalal has noted, every tracker fails eventually and so failure detection of the tracker is paramount to the success of the algorithm. The failure heuristic is related to the Median Absolute Deviation of the pixels within the region of interest. From Dr. Kalel's thesis:

> "Let δ_i denote the displacement of a single point of the Median-Flow tracker and δ_m be the displacement of all points. A failure of the tracker is declared is Median Absolute Deviation (MAD) is larger than a threshold, median ($|\delta_i - \delta_m|$)> 10 pixels. This heuristic is able to reliably identify most failure caused by fast motion of fast occlusion of the object of interest. In that case, the individual displacement become scattered around the image and the MAD rapidly increases." (Kalal [30])

In summary, the tracking component of OpenTLD is robust, yet like all trackers will fail given time or circumstance. All trackers are subject to drift due to environmental conditions or significant change in the track object's profile. That is, changes in the environment of the image (i.e. lighting, contrast) or deviations from the tracker's current model of the object. This is why the learning component of the OpenTLD algorithm is paramount to its performance.

2.7.2.2 Learning

The primary tool cultivated by OpenTLD for learning is P-N learning. "P" stands for Positive and "N" stands for negative respectively. These are represented as the "P-expert" and the "N-expert" as displayed in Figure 13. The goal of the P-expert is to determine whether a part of the image is, in fact, part of the object of interest. If deemed so, the P-expert updates the object training model and increases the generality of the detector (Kalal [30]). Conversely, the N-expert provides the detector with negative training examples and therefore helps to determine what is not the object of interest. The key assumption made by the N-expert that is not made by the P-expert is that the object of interest can occupy at most one location in the image at any time (Kalal [30]).

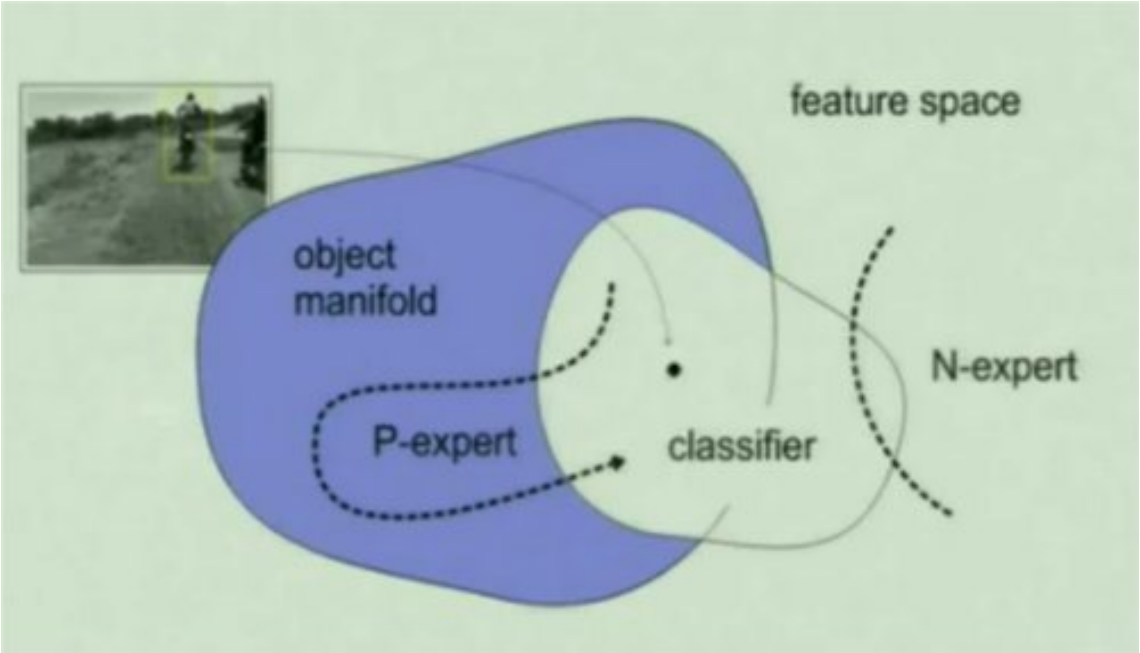


Figure 13: P-N learning implemented in OpenTLD

2.7.2.3 Detection

From the data provided by the learning and tracking elements, classifiers are continually trained for each image in the video stream. These classifiers consist of three stages as seen in Figure 14:

1. Patch variance
2. Ensemble classifier
3. Nearest neighbor

Each stage will either reject the region of interest or pass it on to the next classifier for processing (Kalal [30]).

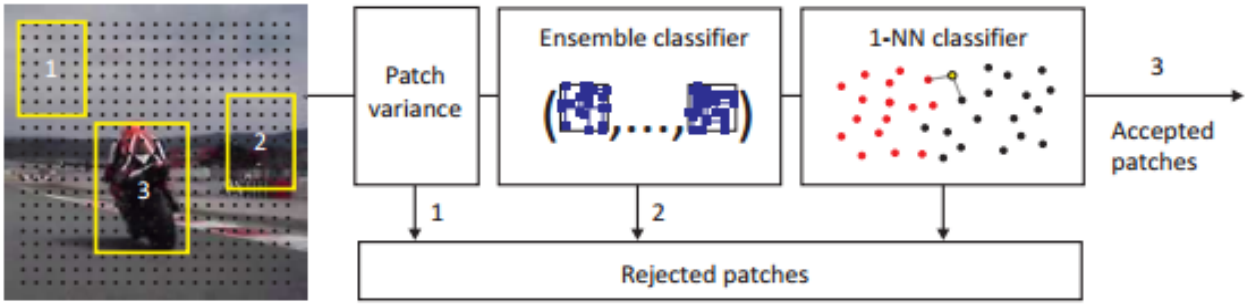


Figure 14: Block diagram of the detector's three stages

The patch variance stages serves to cull a large portion of the image quickly, and takes advantage of the fact that generally there will be a sizable section of the image that is not the object of interest. An example of areas that the patch variance detector would commonly cull is portion of the sky in an outdoor image, where it is composed of primarily one color. The ensemble classier serves to further refine the region of interest by analyzing the patches passed through the patch classifier, and so to the nearest neighbor classifier. Each classifier essentially analyzes a smaller portion of the image acting like multiple sieves. This creates an efficient process whereby computational resources are not wasted on fine analysis of what is obviously not the object of interest.

2.7.2.4 Conclusion

In summary, the OpenTLD algorithm is ideal for our application given these factors:

1. Versatility
2. Robustness
3. Efficiency

The tracker is versatile in that it may track any object with the only initialization being a region of an image defined by the user. This is ideal for our application in that the DIVER ROV will be called upon to track different objects in numerous environments. Objects that the DIVER ROV may need to track include scuba divers, marine life, ocean vessels, etc. Environments may include clear water, murky water, open water and cluttered lake bottoms. These are the conditions in which the robot will be called upon to operate in. Conditions in which all previously discussed algorithms might succeed in one case but would fail in the others. Building upon versatility, robustness is essential to the DIVER ROV's operational capability. Part of the operational goals of the robot can include monitoring dangerous conditions. If the robot loses tracking at a sensitive or critical time, it may mean injury or death for a diver. OpenTLD's adaptability makes the algorithm robust, and therefore may end up avoiding costly consequences. Finally OpenTLD is efficient. As the command and control software may be run on any surface station capable of supporting ROS, the tracker may be called upon to operate on substandard hardware. Our empirical testing shows that the tracker is capable of running on extremely limited hardware and still produce a usable result.

CHAPTER 3: The Dangerous Inspection & Versatile Exploration Robot

3. Introduction

Chapter 3 discusses the methods and processes for the design, manufacture and testing for the DIVER ROV. This chapter will cover the integration of commercial and custom components and the engineering processes for selection for each component. Specifically Chapter 3 introduces the DIVER ROV concept, discusses the design and fabrication of the frame and thrusters, describes the software and programming utilized for the DIVER ROV and provides an in-depth explanation of the electronic sensor and hardware used.

3.1 The DIVER ROV Concept

The DIVER ROV system concept was designed for direct integration with the professional diving community. The purpose of this system is to assist track and monitor professional scuba divers. For safety purposes, typically multiple divers are sent to investigate and react to any given underwater scenario. Diving teams are deployed for repair, salvage and rescue applications in dangerous conditions and locations. The DIVER ROV uses tracking software to provide surface operators with the necessary information to protect these professional diving teams. A high resolution forward facing camera on the ROV allows the surface operator to visualize the mission in real time. The surface operator can respond quickly and effectively to life threatening scenarios and dispatch the appropriate commands and personnel through the use of LED sequenced Morse code. By facilitating communication, surface operators are able to assist and monitor in underwater missions through the use of the DIVER ROV. The features of the DIVER ROV system allow for seamless integration with any professional diving team, improving the overall safety of the team.

3.1.1 Design Specifications

As part of the design stage, The DIVER ROV team developed specifications to aid the development of a platform capable of the missions specified under the purpose statement. The DIVER ROV must be capable of assisting, tracking and monitoring professional scuba divers,

providing the DIVER team with a mission statement to develop design specifications. The DIVER ROV must be capable of operating in missions where professional divers are present. The design of the ROV allows for depths and temperatures that professional divers would experience. For the proof of concept of the DIVER ROV, the team decided to create an operating conditions defined by a depth between 0 and 50ft and water temperature between 50 to 70F. Defining the mission specifications allowed the team to create and select components that were viable in this operating range. The engineering team was tasked with the selection of materials and commercial components able to withstand wide ranges of pressures and temperatures while also meeting budget needs and waterproofing requirements. Manufacturer's specifications allow the team to select the proper materials and components able to increase the functionality of the DIVER ROV. In order to meet the design criteria, weight and size considerations were made allowing for seamless integration of the ROV with scuba teams. By providing a compact, lightweight and highly maneuverable ROV, the team was able to remain true to the mission statement of the DIVER ROV.

3.2 Frame

The DIVER ROV is designed to hydrodynamic, durable and lightweight. The frame of the system is created using sheets of three eighths inch thick acrylic. Acrylic is relatively inexpensive and easy to manufacture. Rather than writing and executing complex machining code for a computer driven mill, the team was able to easily cut the desired shape and specifications with minimal machine programming, utilizing a laser cutter in the process. Major frame components have slotted features that allow for simple assembly using plastic cement rather than a series of complex holes and hardware. This assembly method reduces error during assembly and can help extend the life of the ROV frame by reducing the amount of metal components which can be susceptible to corrosion.

3.2.1 Frame Design

The design of the frame features a rounded front and rear section as well as six supporting cross beams. The frame consists of three major components, the main side runner, primary vertical support beam and secondary angular support beam. These frame components provide ample structural support for the remotely operated vehicle (ROV) for compressive and tensile stresses in all three dimensions. Two side frame runners form the major components for the frame assembly. The frame side runner, depicted in Figure 15, provides the base mounting surface for the internal frame supports. Rectangular laser cut slots are cut into both of the frame components to provide locations for the internal supports to slide directly into. Two central cross beams, seen in Figure 16, are placed vertically to support the center of the assembly and provide a mounting surface for the central thruster. Four additional beams are placed in angular slots to prevent the frame from collapsing and to provide additional impact resistance. These supports, seen in Figure 17, are smaller in size and serve to reinforce the front and rear sections of the frame, protecting vital electronic components, such as cameras, from damage in the event of a collision. Rear supports provide support for the tether by allowing for mounting areas to disperse stresses from the tether onto the frame of the ROV. Once these components are fit together, epoxy is applied at each location to prevent frame failure by disassembly.

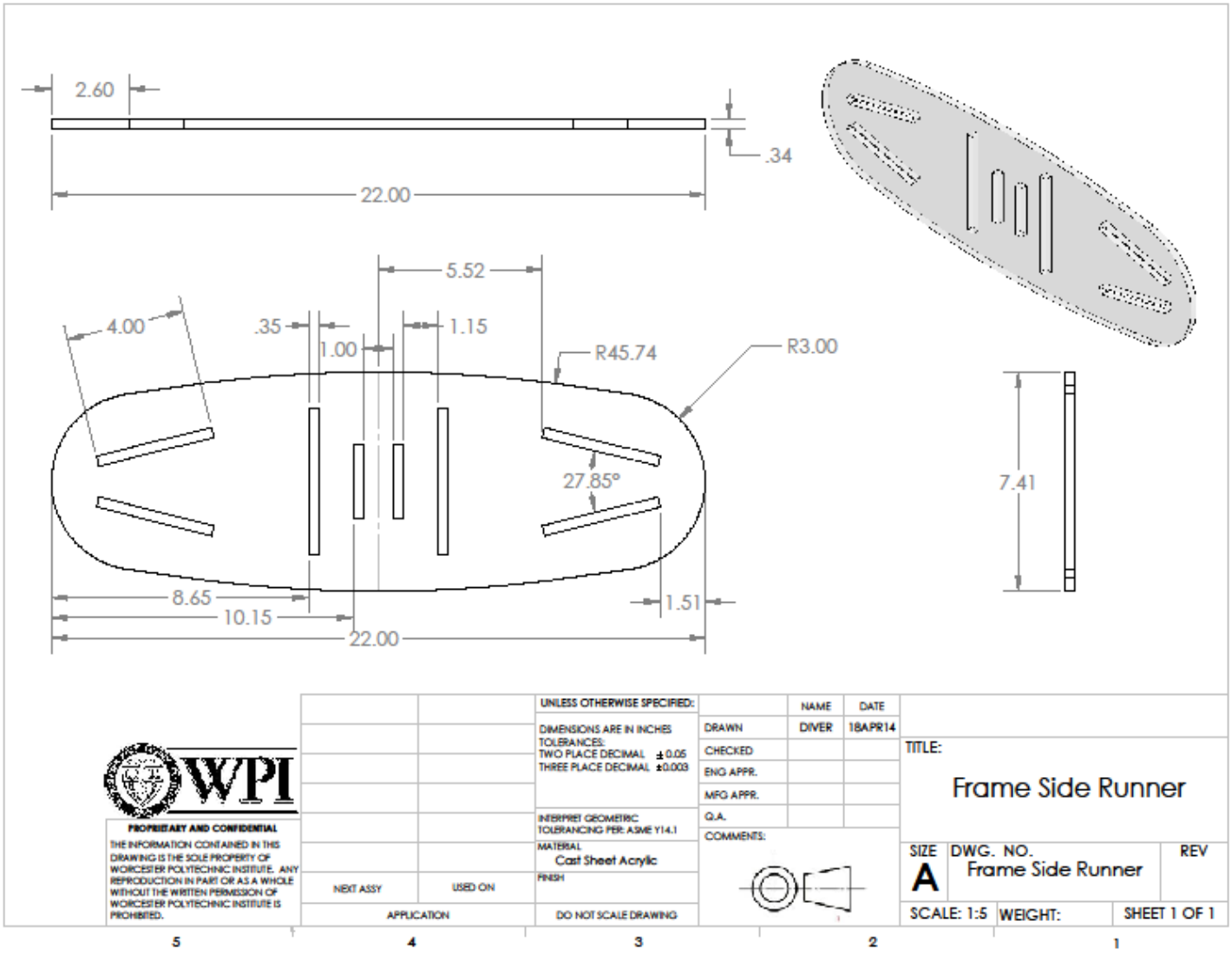


Figure 15: Side Runner Drawing

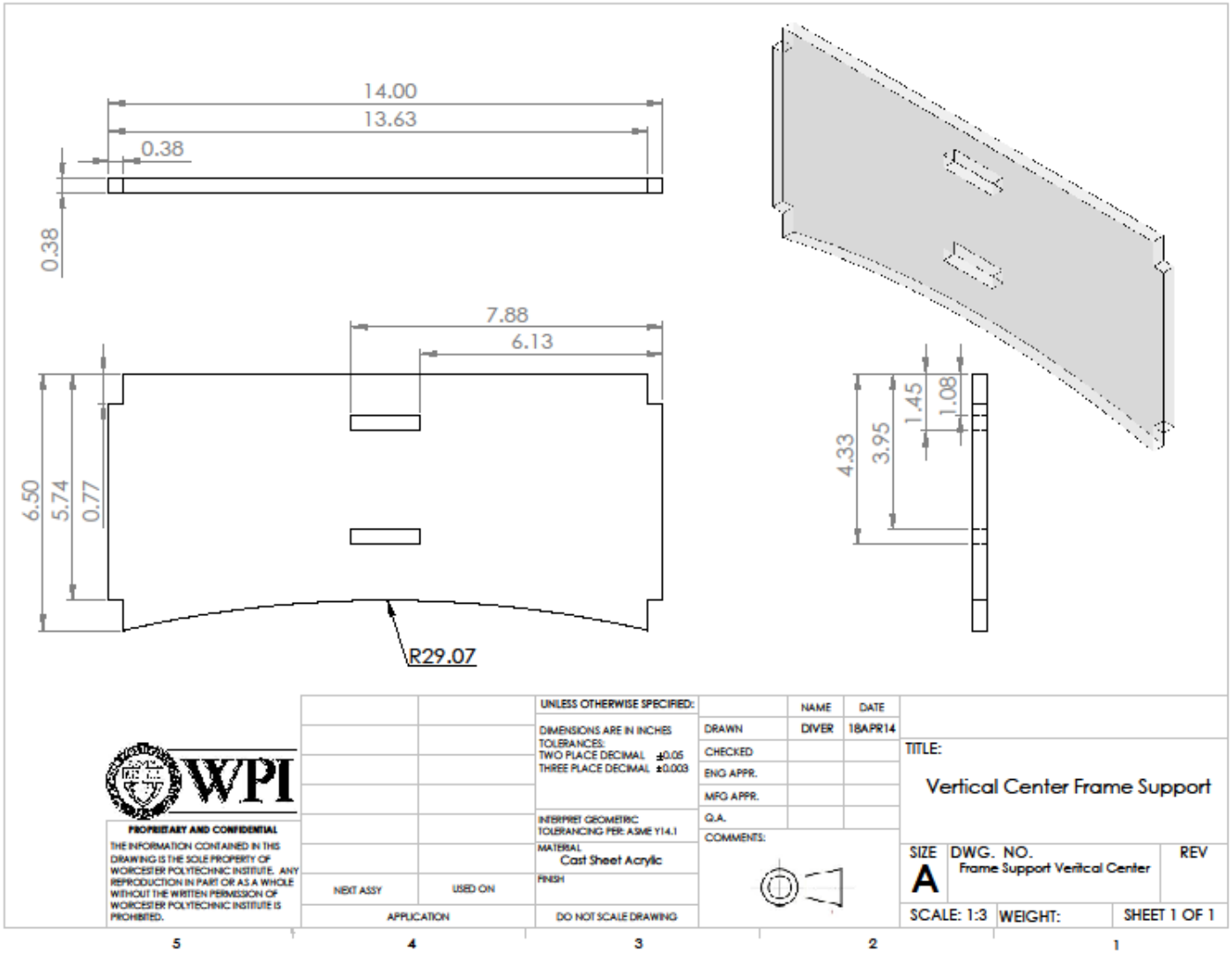


Figure 16: Vertical Support Drawing

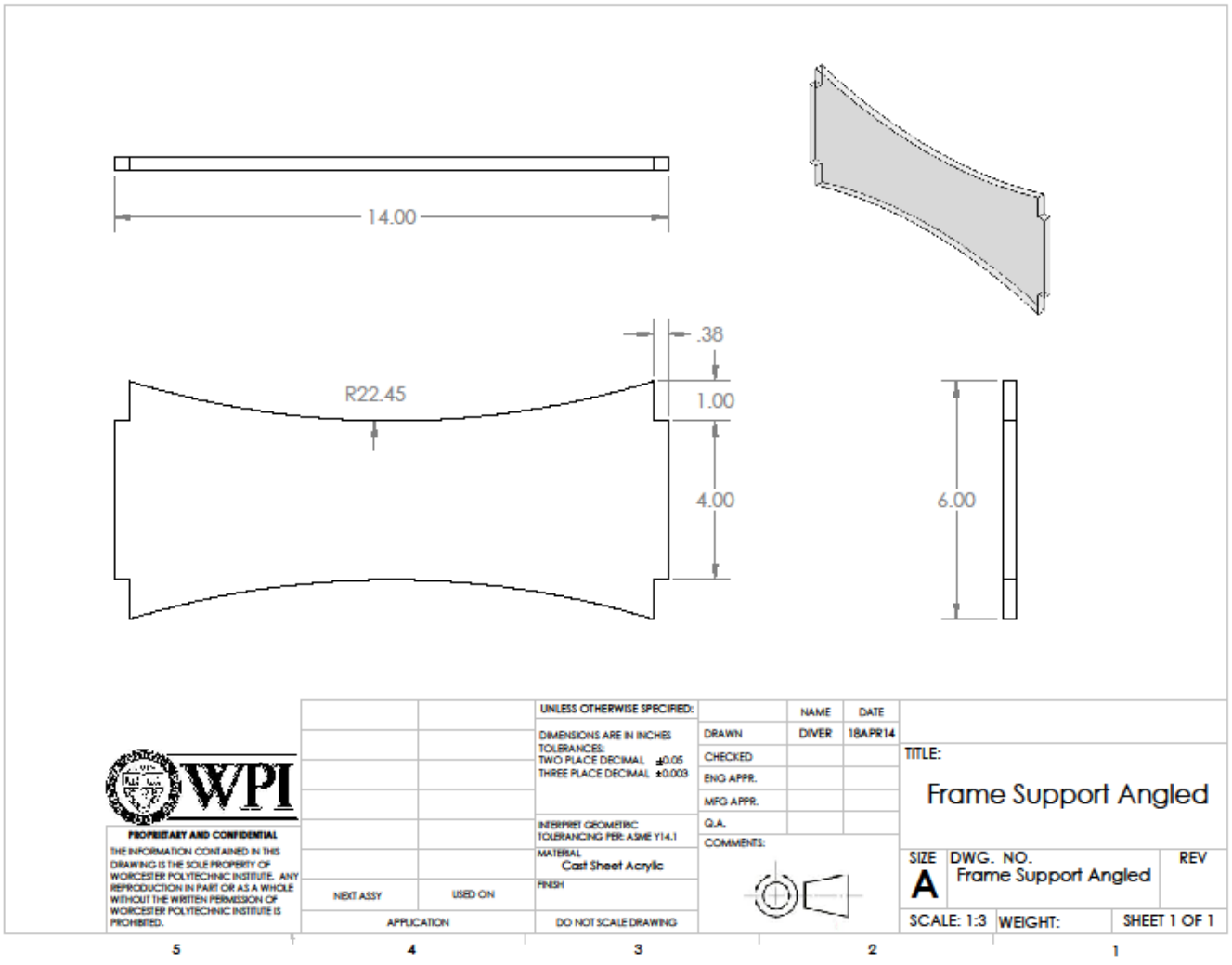


Figure 17: Angled Support Drawing

The thruster mounting assemblies are designed to be similar in processing and assembly to the main frame. There are two different types of mounting assemblies based on the desired location of the thruster. The primary drive thrusters are mounted externally on the side of the frame runners while the internal surface dive thruster is mounted internally to the frame. The external drive thrusters are mounted using triangular mounting brackets. The bracket holds the thruster at the desired distance away from the frame to allow for maximum flow of water for maximum forward thrust. This bracket can be mounted to the assigned slot feature on the side runner of the frame and epoxied in place. Maintaining a constant orientation for the thruster is crucial to mission viability making the mounting bracket for this component vital. Similarly, the internal mounting bracket must maintain constant position and orientation to allow this component to function properly. Figure 18 shows the SolidWorks model of the frame assembly.

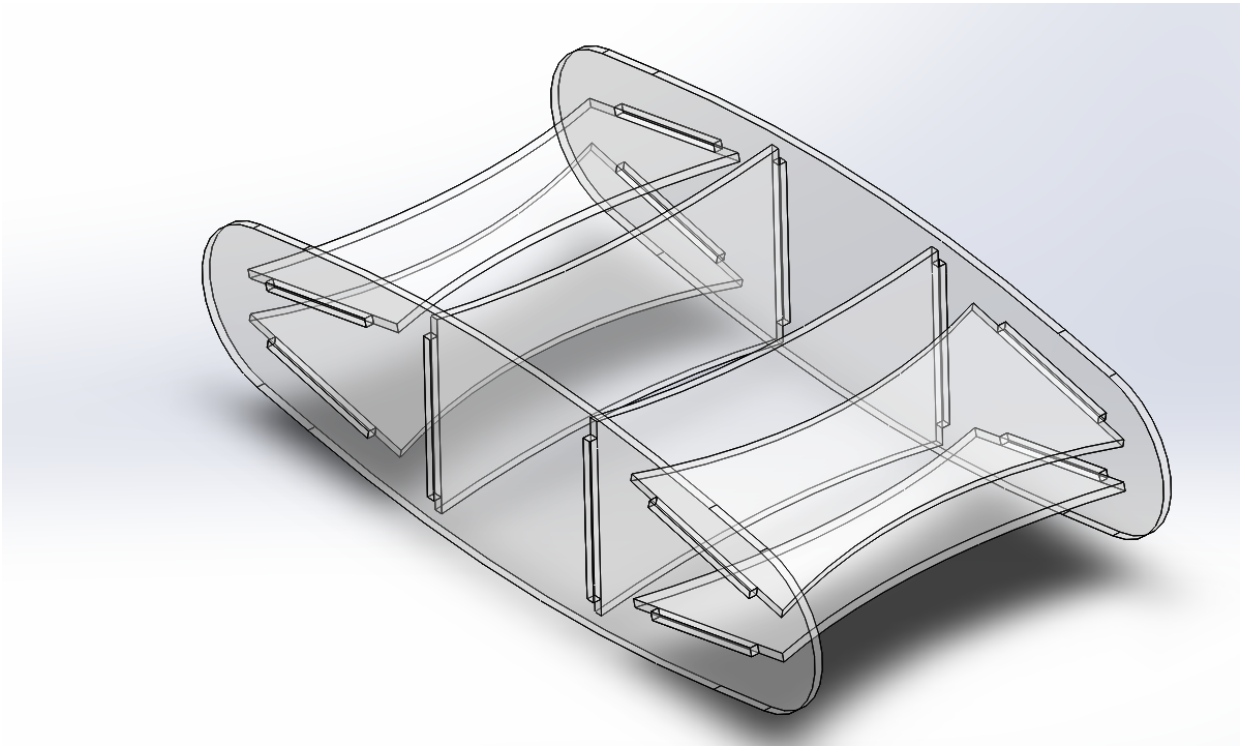


Figure 18: Frame Assembly Model

The frame serves to provide structural support for mechanical systems and the protection and housing for the electrical components. The front and rear mounted electronics housing allows the remotely operated vehicle to sustain collisions while remaining operational. These compartments house the Arduino microcontroller, Raspberry Pi and each of the motor drivers. Mounting this compartment internally to the ROV allows for easier access to the tethering system by significantly reducing the amount of internal cabling and waterproofing that the ROV would require if the housing was front mounted. The frame design also provides a protected front pocket for all cameras and sensors, limiting the amount of debris that will contact these vital electronic components.

As a whole the frame design provides the vital stability for the long term functionality and mission success of the DIVER ROV. The system is designed to sustain higher pressures associated with underwater missions at depths greater that thirty three feet below sea level, as well as the sudden impacts with unexpected objects. The frame will provide the strength to allow the operator to complete each mission scenario that the DIVER ROV is presented with. Each time the system is placed alongside the professional dive team, the ROV will be able to provide vital information from surface operators and divers alike, improving the overall mission safety in any scenario.

3.2.1.1 Elliptic Airfoil Design

For the DIVER ROV, an elliptical shape was chosen as the primary design for the ROV hull. This modified hydrofoil shape was chosen because of the hydrodynamic effectiveness of an airfoil and the ability of an ellipse to travel in reverse. The purpose of the ROV requires it to have the ability to maneuver in either direction when being controlled by the surface. In autonomous mode, the ROV must be able to move fluidly while following the diver.

When designing the ROV, two main factors were taken into consideration, stability and buoyancy. That is, it cannot tip over and it doesn't sink to the bottom, but it also does not float on the top of the water. Buoyancy and stability underwater will increase the efficiency of the ROV by decreasing the energy required for movement.

The DIVER ROV is designed with an open hull. The elliptic portion refers to the side panels of the ROV. These are designed as elliptic airfoils to give the ROV the buoyancy and stability mentioned above. The two elliptic side panels have a slotted design to allow support panels to connect them and allow other components to be placed on the supports between the side panels. The open system allows the various components, such as the camera and electronics housings, to be easily accessible between missions. The open hull also allows the ROV to be as light as possible by minimizing material while the elliptical side panels maintain its hydrodynamic intent.

The simple ellipse shape also allows the main thrusters to be placed on the sides where they will be most effective in controlling forward and aft thrust as well as yaw of the ROV. The design also allows the placement a thruster directly in the center of the ROV to control up and down motion of the vehicle.

3.2.1.2 Hydrodynamic Analysis

Calculating the drag on an underwater ROV is important because it allows you to calculate the power required to move the ROV underwater. This is a difficult process due to the flooded design of ROV's. As a result, it is easiest to approximate the ROV as a particular solid shape and calculate the drag on that shape. Popular shapes for this approximation include cubes, spheres, and cylinders as shown in Figure 19.

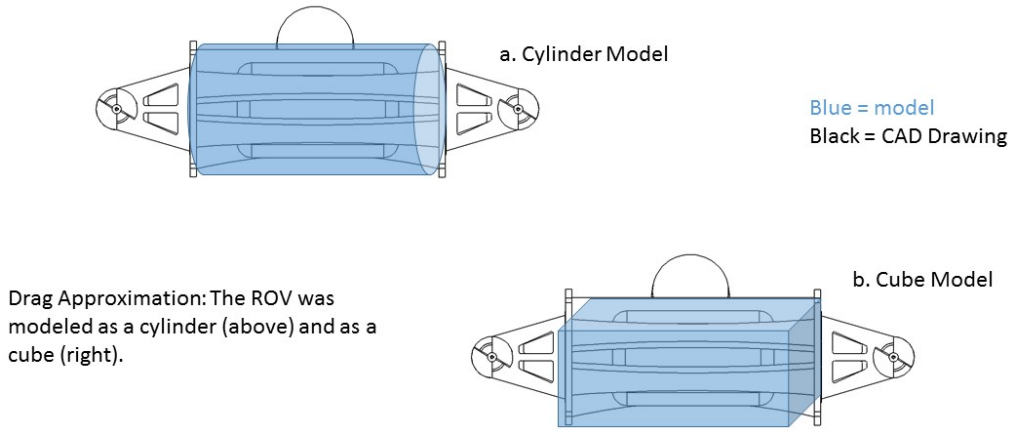


Figure 19: Drag Models for a. Cylinders or b. Cubes

When calculating the drag on a body, you must first find the drag coefficient, C_d. To find the drag coefficient, you need to calculate the Reynolds number. To calculate the Reynolds number you need to know the density of the fluid, the velocity of the body moving through the fluid, the characteristic length of the body, and the viscosity of the fluid. The Reynolds number can be calculated using the following equation: $\mathrm{Re} = \frac{\rho U L}{\mu}$. Once the Reynolds number is calculated and the drag coefficient is found, the next step is to calculate the drag force using the following

equation: $F_D = \frac{1}{2}\rho C_d A U^2$. The ROV in this project is designed to travel through water, therefore the density and viscosity are: $\rho = 10^3 \frac{kg}{m^3}$ and $\mu = 1.002 * 10^{-3} \frac{kg}{m*s}$ respectively. To estimate the drag force, the front view of the ROV was modeled as a cylinder and a cube as shown in Figure 19. In the estimations, the characteristic length, L, is 0.3556 m, the surface area, A, is 0.207 m^2, and desired velocity for testing purposes is 1 meter per second. The above equation is used to calculate the Reynolds number: $Re = 3.549 \text{ x } 10^5$. According to Massey [reference number], the drag coefficient, C_d, for Reynolds numbers between $Re = 2.0 * 10^5$ and $Re = 5.0 * 10^5$ drops from 1.2 to 0.3 (Massey [41]). Using Massey's findings and the Reynold's number calculation, the drag coefficient for the DIVER ROV is assumed to be 0.3.

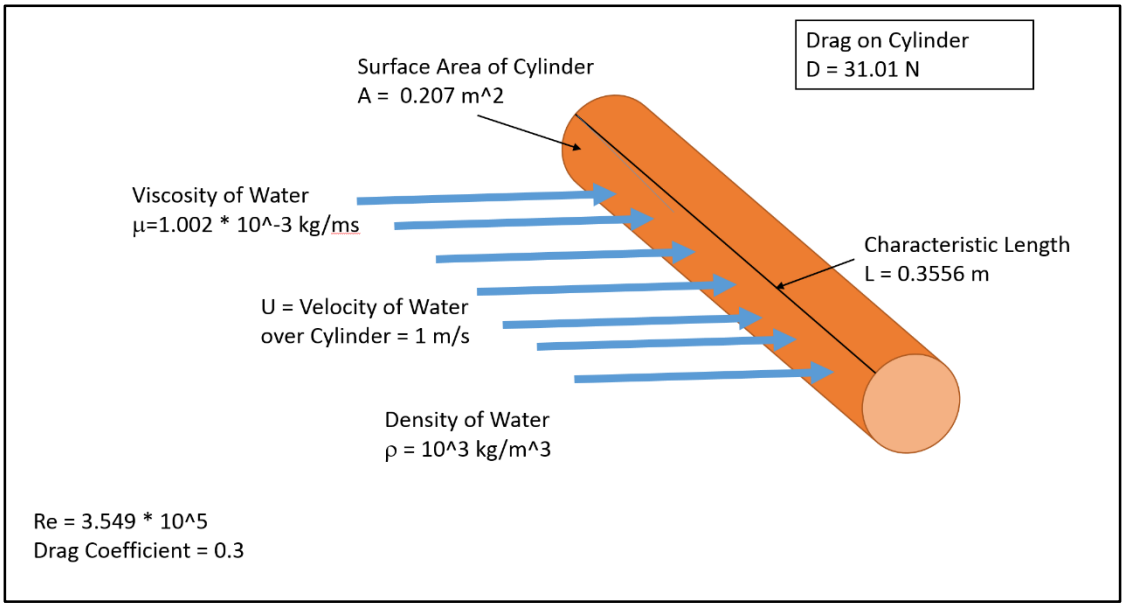


Figure 20: Cylinder Drag Model

The drag force calculated on the cylindrical model of the DIVER ROV, moving through water at 1 meter per second, is 31.01 Newtons, as shown in Figure 20. The Drag force on the cylindrical model is shown in Table 3.

Table 3: Estimated Drag Forces

#	Velocity of Cylinder in Water (m/s)	Estimated Drag Force (N)
1	1	31.01
2	0.5	7.753
3	0.2	1.24

As expected, the drag force decreases with a decrease in velocity underwater. The maximum desired speed of the DIVER ROV is 1 m/s, therefore speeds less than this were used to account for accelerating up to 1 m/s and down to 0 m/s.

Figure 21 compares the estimated drag forces on a cylindrical model of the DIVER ROV with a cubic model of the DIVER ROV. Both the cylindrical and cubic model of the DIVER ROV can be found in Figure 21.

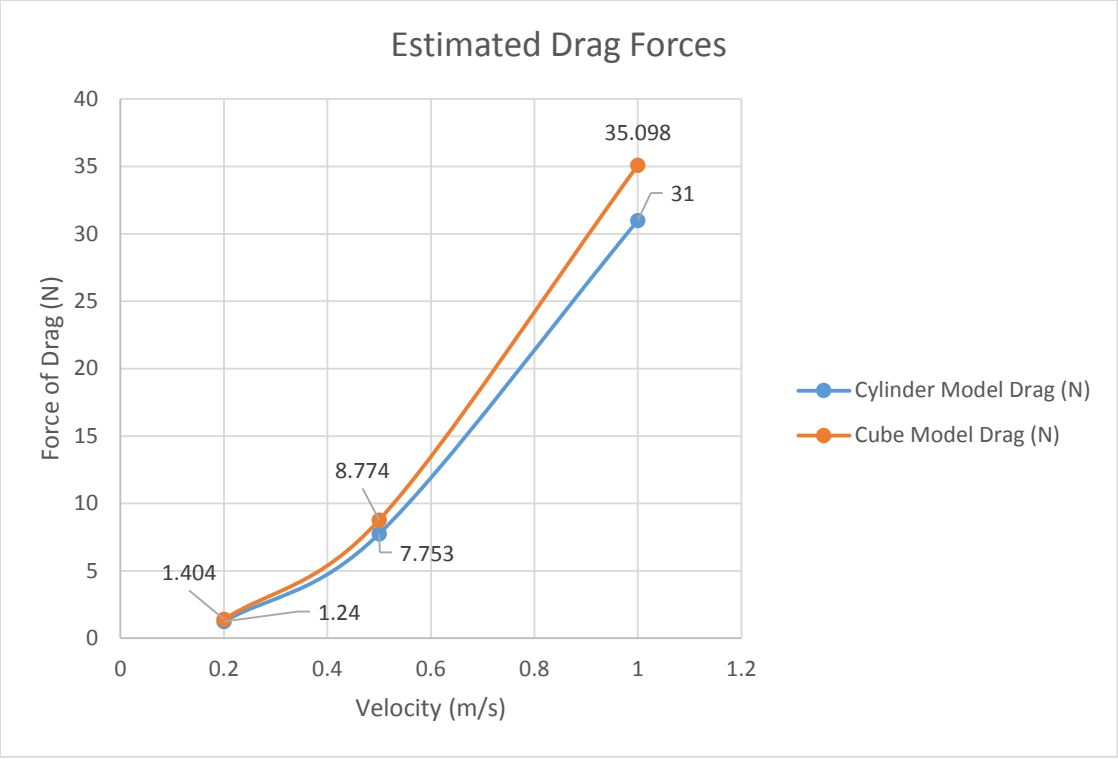


Figure 21: Estimated Drag Forces

The cube model resulted in a higher drag force estimation because of the flat shape of the cube versus the rounded shape of a cylinder. Therefore, using a more cylindrical approach in designing an ROV will result in a system that is more hydrodynamic. The results were used to calculate the power required to overcome the drag on the two models of the DIVER ROV.

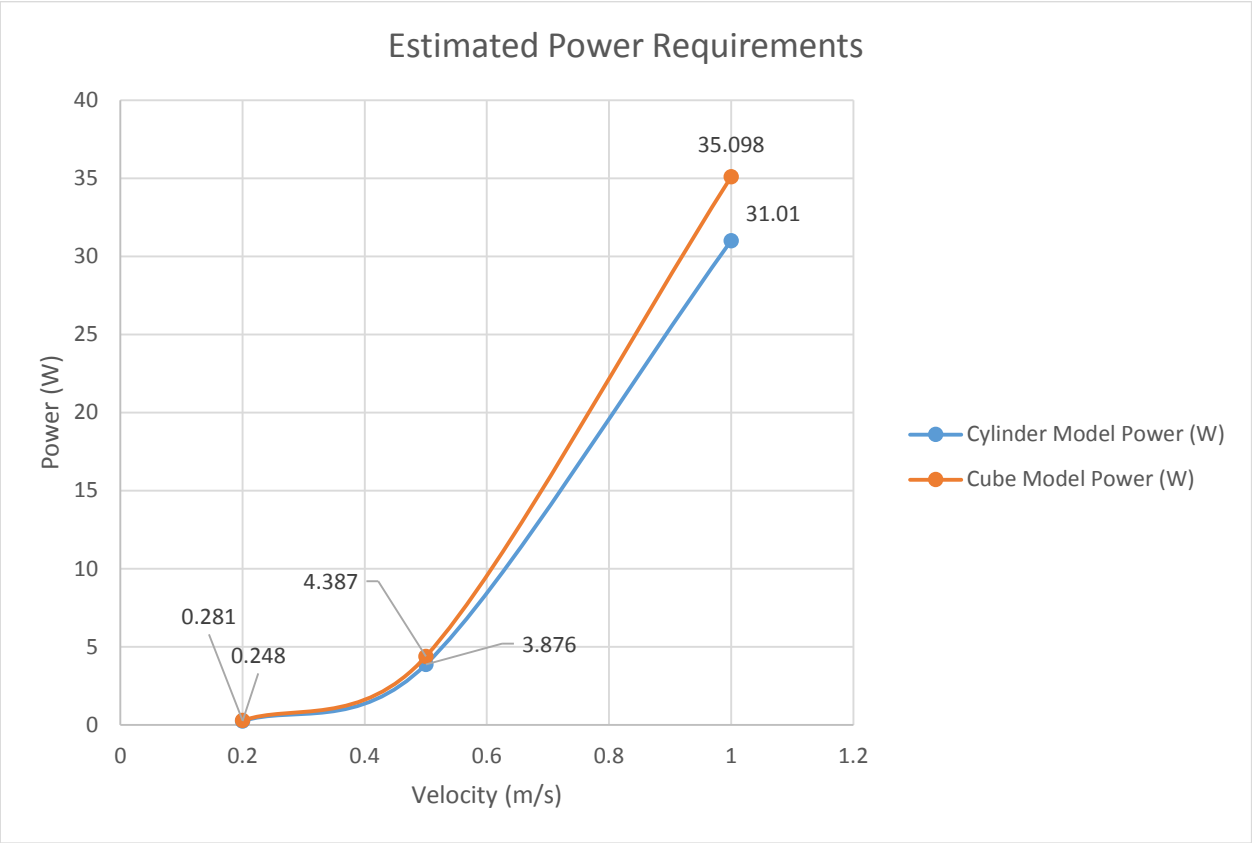


Figure 22: Estimate Power Requirements

Figure 22 shows the results of the power estimations. The graph is similar to that of the drag calculations because of the larger amount of power required to overcome a greater drag force. For example, results show that at a cube with an area of 0.207 m^2, moving through water at 1 m/s would require 35.098 Watts to overcome the drag force due to water.

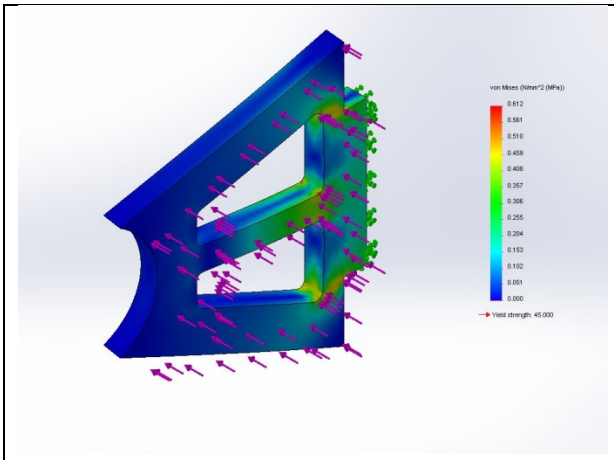
3.2.1.3 Stress Analysis

Analysis of strength of the DIVER ROV frame began with a careful examination of the stresses the robot would be subjected to. The DIVER ROV is exposed to forces resulting from the thrust produced from the bilge motors, forces from operator handling during transportation and use and constant pressure from its aquatic environment.

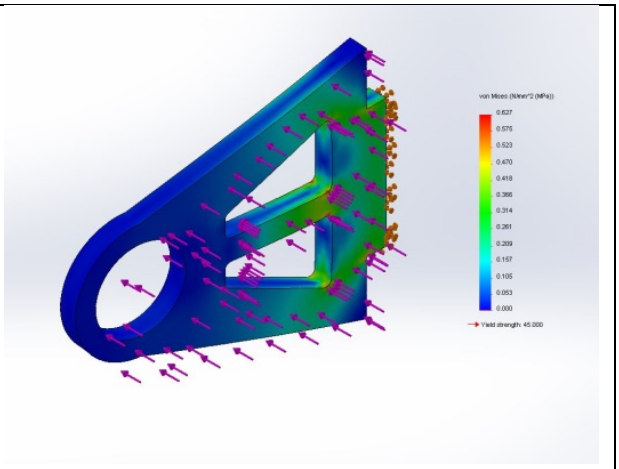
The open frame design of the DIVER ROV allows water to flood the entire frame. This results in water pressure acting uniformly over the entire robot. The acrylic material used in the frame is solid, eliminating any compressible regions from the frame.

While the overall frame is very strong, there are weak points in the design. Specifically, these weak points are the binding points of the horizontal thruster mounts, both front and rear, and the vertical thruster mount. These 3/8" acrylic pieces hold the bilge motors which provide thrust to move the robot. The thrust produced in turn applies stress on the ROV, which centralizes at the thin points of the thruster mounts. Using the SimulationXpress Analysis Wizard component of SolidWorks 2013, an analysis of the Von stresses was conducted. For this analysis, Acrylic (Medium-High Impact) was selected with a Yield Strength of $4.5x10^7 \ \frac{N}{m^2}$ and an Elastic Modulus of $3.0x10^9 \ \frac{N}{m^2}$.

Model propellers like the ones used in the DIVER ROV output between .8 and 2.2 N of force (Thone [68]). Since an exact output force for our thruster is not available, we ran the stress analysis using 5 Newtons of force. This incorporates a 2 to 3 factor of safety. In addition, both the Front Thruster Mount and the Rear Thruster Mount were analyzed separately and with the full 5 Newton of force. In reality, the force produced from the thruster will be split between the two mounts. Figure 23 a) below shows the von Mises stress diagram for the Front Thruster Mount while Figure 23 b) shows the von Mises stress diagram for the Rear Thruster Mount.



a) Analysis of Front Thruster Mount



b) Analysis of Rear Thruster Mount

Figure 23: Stress Analysis of Thruster Mounts

As seen in Figure 23 a) and Figure 23 b), the front and rear thrusters each withstood the 5 Newtons of applied force. As suspected, the areas of highest stress are narrow spots at the base of the mounts.

The Vertical Thruster Mount is subject to the same force as the Front and Rear Thruster Mounts but is secured at two points instead of one. As with the Front and Rear Thruster Mounts, a stress analysis was conducted using the SimulationXpress Analysis Wizard component of SolidWorks 2013. 5 Newtons was chosen as the applied stress. The von Mises stress results can be seen in Figure 24.

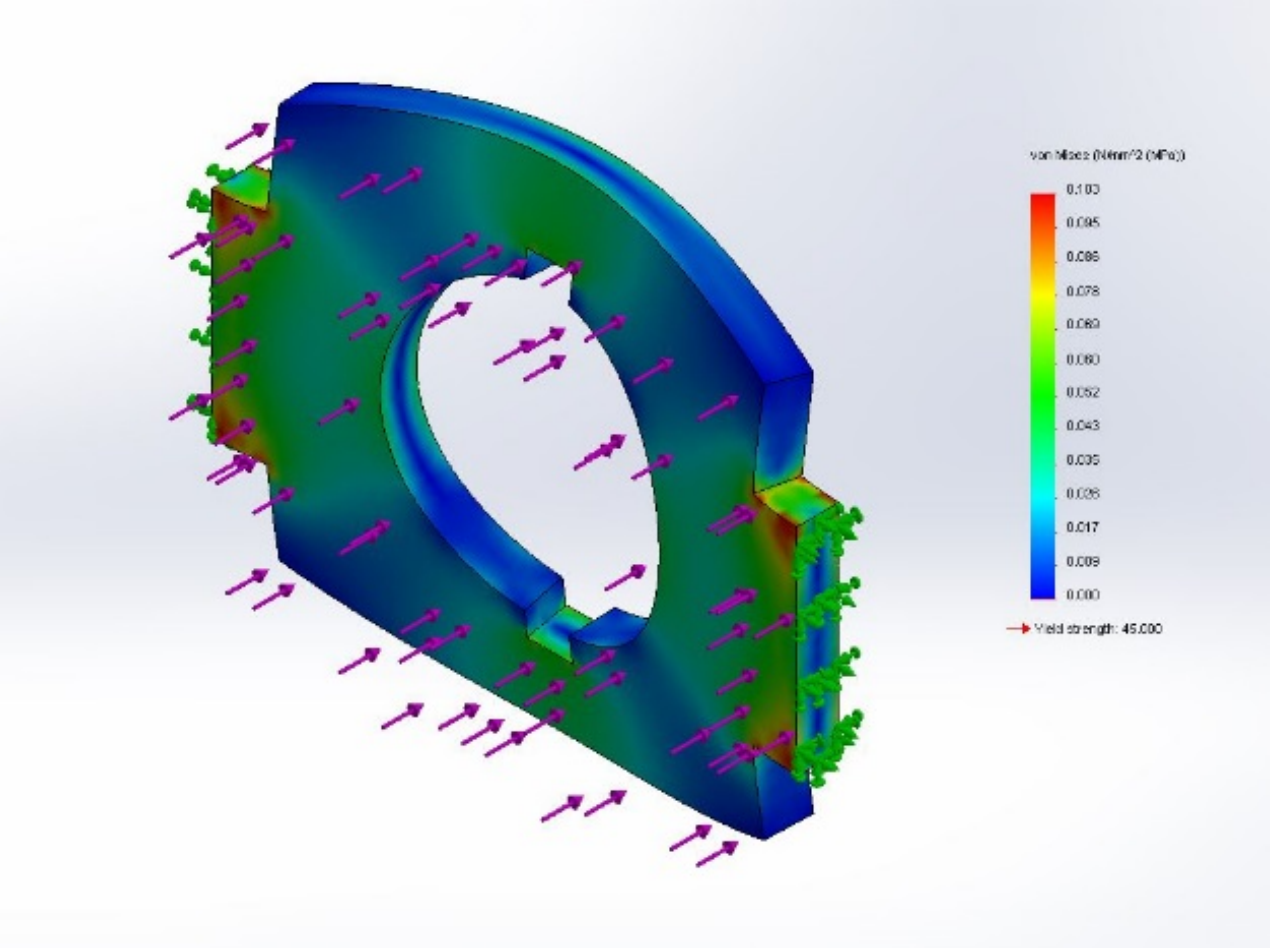


Figure 24: Vertical Thruster Mount Stress Analysis

The results show that the Vertical Thruster Mount can withstand the 5 Newtons. In addition, Figure 24 shows that the areas of highest stress are at the two anchoring points of the piece.

By analyzing the most at risk components of the DIVER ROV, we ensure that it can withstand the thrust forces of normal usage.

3.2.2 Material Selection

The material selection process for the DIVER ROV began with a close examination of the system requirements we established for the project. The ROV needed to be rigid and strong enough to function properly yet small and light enough to be versatile and easily deployable. It needed to be corrosion resistant to withstand an aquatic environment in which it functions and durable enough to withstand transportation and storage.

The main goal of this ROV is a proof of concept. Because of this, certain factors needed to be taken into account. First and foremost, there was a limited budget. With the electronic components making up the majority of the budget, the amount allocated to materials was stringently small. Secondly, since the application is solely experimental, durability can be sacrificed. In other words, this ROV is not designed to be ready to be utilized in commercial applications and therefore does not have to be as durable or user friendly as a product on the market. This gave us more options for materials.

We considered a wide range of materials in the design and manufacture of the DIVER ROV. Steel, while strong, inexpensive and easy to manufacture, was too heavy and would have required measures to make it resist corrosion. Aluminum is far lighter than steel but costs more and is more difficult to manufacture. Polymers offered decent strength and good manufacturability at an affordable price and light weight. It was for these reasons we chose to build the frame out of polymer.

Choosing a polymer proved to be a long, drawn out decision process. Acrylic is an extremely common plastic, readily available in sheets and bars of various lengths, widths and thicknesses. It is easy to process, as it can be milled and turned or cut from a laser cutter. Its only downfall is its brittleness. Acrylic has poor plastic deformation properties, snapping and shattering upon failing. Polycarbonate, an alternative polymer, is far more durable than acrylic. While harder

to find than acrylic, polycarbonate is available in the sizes we needed but for a higher price. In addition, polycarbonate must be milled or turned, as it cannot be laser cut in the WPI facilities.

Quite simply, the argument for using acrylic outweighs that of using polycarbonate. Using acrylic will save us both time and money. Acrylic costs less than polycarbonate and by laser cutting instead of milling and turning we can produce the same product in less than half the time, not including the fact that it will require a fraction of the programming. While durability is a concern, the DIVER ROV is a test robot and will not be subject to abuse of a product used in commercial or industrial conditions. Acrylic sets easily with the proper bonding cement and is overall the smarter material to use.

3.2.2.1 Poly Methyl Methacrylate (PMMA)

The team chose to create the main frame of the DIVER ROV out of cast Poly (methyl methacrylate) or PMMA, commonly referred to as acrylic. This material is easily machine able, high strength, and relatively low cost. Sheets of standard acrylic can cut using a high powered laser without risk of deformation from melting or ignition while processing. Processing acrylic from laser cutting sheets reduced the amount of machining time required and the complexity of the manufacturing process. Acrylic has an overall tensile strength between 48 and 70 MPa, determined by the number of syndiotactic polymer chains found in the material. Standard commercial grades of acrylic contain between 50 and 70% syndiotactic polymer chains, increasing the overall strength of the material by improving intermolecular bonds and crystallinity. Acrylic was selected as a low cost and low weight alternative for the DIVER ROV system. The material can be easily improved through the application of fiberglass to reduce fracture during impact.

3.2.2.2 Fiberglass

With brittleness being the biggest disadvantage of acrylic polymer, we decided to reinforce the frame pieces with Fiberglass composite. This decision was made after breaking on of the side runner frame pieces during the initial fabrication. By adding a layer of fiberglass composite to each side of all frame pieces, the overall tensile strength of the piece is raised, without drastically increasing weight. This ensures that the frame can bare the constant loading forces applied to it such as thrust from the motors and withstand the stresses involved in underwater impacts with objects the ROV may encounter during operation. Figure 25 depicts a layer of fiberglass prior to the addition of epoxy ("Boat" [5]).

Figure 25: Fiberglass Mesh

3.2.3 Frame Manufacture and Assembly

The design process was initially conceptualized through researching commercially sold and home build remotely operated vehicles (ROVs). Through this research, the team was able to understand the basic concepts important for the design of an underwater robotic system. After an initial concept was determined, hand sketches were produced. The viability of these hand sketches was determined using basic stress, strain and fluid dynamic analysis and a refined design was determined. Once the refined concept was produced, components were modeled through the use of computer aided modeling software. The design team chose to use the commonly used program SolidWorks because of its use in professional engineering as well as its available functionality for calculating more advanced stress and strain concentrations as well as fluid analysis.

During the design process, considerations were given to manufacturability of each component. Because of the specificity of the mission parameters, it was necessary to create the frame from scratch. The team met with machinists to determine the ideal process for creating each of the specialized components in the DIVER ROV. During these meetings, the team determined that the ideal machining process would be laser cutting— a process in which components are cut from flat sheets of material using a high intensity laser beam. This processing technique allowed the frame to be produced from SolidWorks files rather than convert the data into machine code for use in the computer driven mini-mill. Figure 26 below shows the manufacture of the frame components on the Laser Cutter.

Figure 26: Laser Cutting the Frame

Once the design and processing techniques had been determined, the team selected the ideal component thickness for the frame based on material cost, desired strength, and limitations of the laser cutting process. Laser cutting involves a high intensity laser that is directly related to the thickness of the material. Thicker materials cause a larger temperature difference in the material, leading to surface melting and fire which can destroy the component. After careful consideration, the team determined that the ideal thickness for the material was three eighths of an inch. At this thickness there would be little risk of melting during the cutting process while also allowing the frame design to retain structural integrity.

Throughout the machining process, the team experienced manufacturing setbacks due to tolerance differences in the cast PMMA sheets. While the raw material sheets were considered to be within the manufacturers specifications for tolerances, the interference fit components on the DIVER ROV frame were not to proper specification. The manufacturing team made adjustments

to the SolidWorks files and re-machined components that had failed in the initial laser cutting. Figure 27 below shows the completed frame.

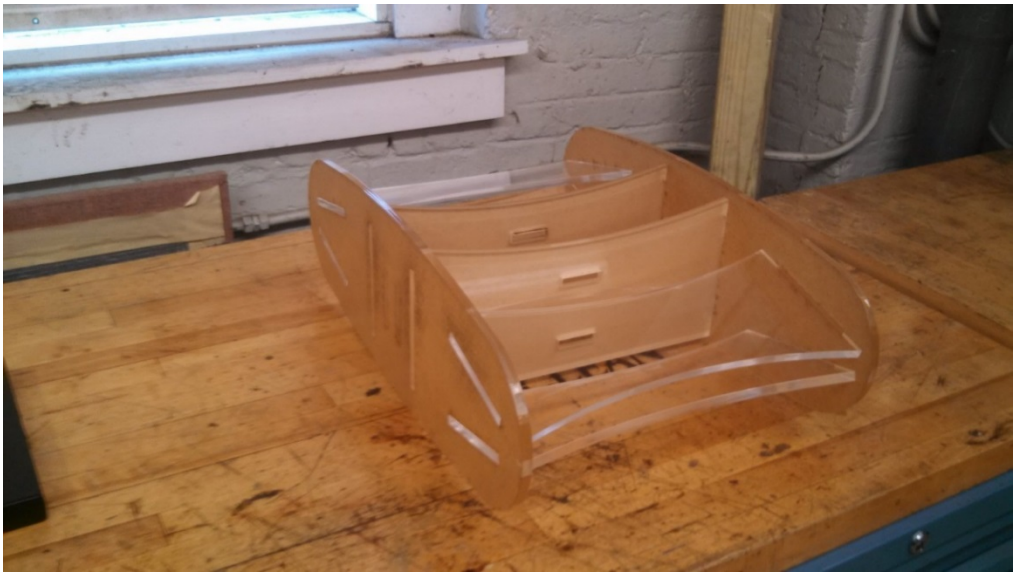


Figure 27: Cut Frame Components

To reinforce the acrylic pieces for strength and durability, each component of the frame was reinforced with a layer of fiberglass composite. Figure 28 below shows the process of reinforcing the DIVER ROV components with fiberglass composite.

Figure 28: Fiberglass Reinforcement

3.3 Propulsion

One major component in the design of the DIVER ROV is the propulsion system. This system allows for the mobility of the robot to be achieved. Careful consideration for the output of each component of this system must be considered. The thrusters, propellers and mounting locations and structures must all be analyzed to provide the desired top speed and maneuverability while maintaining low power consumption. Stress concentrations must be calculated to determine to ensure that the frame will not fail under operation or produce enough drag to cancel the forward motion of the robot. Each of these topics are elaborated upon in the sections below, providing detailed descriptions of the thrusters, propellers, and mounting frame components.

3.3.1 Thruster Research and Design

Thrusters are an essential aspect of any underwater vehicle. For the DIVER ROV, industrial thrusters were included in the original design. The industrial thrusters would provide the most efficient and effective means of maneuver for the DIVER ROV. These thrusters include all the necessary wiring and hardware for the ROV. The quality of industrial thrusters varies, as well as the price. However, the price was too high to fit the budget of this project. After contacting Tecnadyne Inc. as well as Robomarine, there was a need to pursue other options due to the cost. Averaging $3,000, the industrial thrusters were far outside our budget of $160 per student.

3.3.2 Fabricated Thrusters from Bilge Motors

Once we determined that commercial thrusters were not economical for the scope of this project, we researched methods to fabricate our own thrusters. A common method among ROV hobbyist's for fabricating thrusters is to use bilge pump motors. Bilge Pumps are used to remove water from the bottom of a ship's hull, known as a bilge. They are categorized by the rate at which they can pump water and common sizes include 500, 750 and 1100 gallons per hour (GPH) for recreational boats. Due to the nature of this application, bilge pumps contain small, yet powerful motors. Most importantly, these motors are sealed and fully waterproof. Most designs utilize an

impeller style method of function. For the purpose of creating a thruster for an underwater ROV, we are only concerned with the motor itself.

Two 1100 GPH motors were selected to form the two side thrusters and one 1000 GPH motor was selected to form the vertical thruster. Commercial thrusters incorporate a carefully engineered shroud to increase flow rate and thrust. In addition, these shrouds protect the thruster's propeller and prevent debris from easily entangling the device. Unfortunately, unless the thruster's shroud is precisely engineered, it will actually decrease the efficiency of the thruster. No shroud is more efficient than a faulty one. Therefore, given the scope of this project, we decided to not incorporate shrouds onto our thrusters.

3.3.3 Propellers

The DIVER ROV features three individual thrusters, each with an identical propeller. This design consideration allows for each propeller to have uniform performance while operating. This will allow the operator to have more uniform control of the ROV despite external water conditions. Prior knowledge of recreational boating and propeller performance allow the team understand the selection criteria for a propeller and select the best propeller for the DIVER ROV. Figure 29 illustrates some of the considerations to adhere to when selecting a propeller ("Boat Propeller [6]).

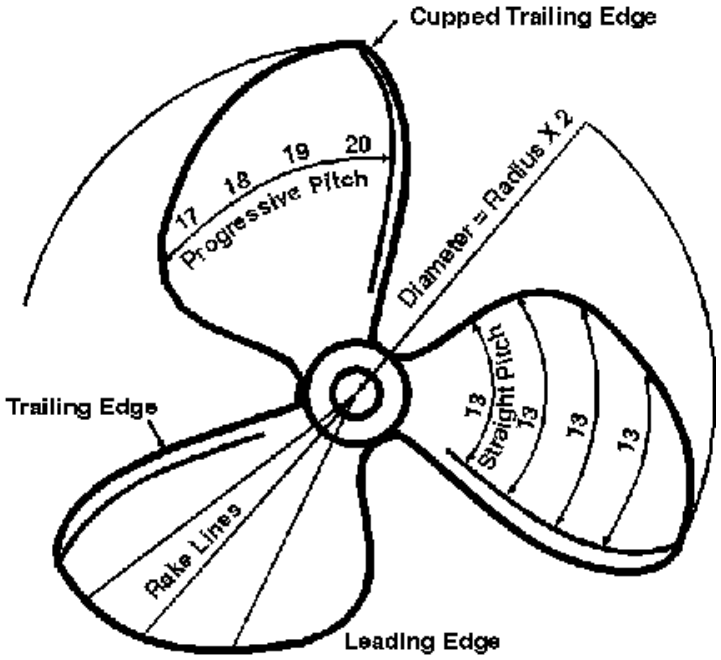


Figure 29: Propeller Diagram

For the DIVER ROV, the team selected a propeller with consideration for performance, speed, weight and size of the DIVER ROV. Considerations were made based of knowledge of recreational boating and availability of small scale propellers capable of integration with the DIVER ROV. The selected propeller features two blades, a small diameter, a high rake angle and nylon polymer construction. Two bladed propellers were chosen because of size of the ROV and the overall cost of the propeller. The smaller diameter propeller was also selected to avoid collision with any of the frame components and to provide maximum thrust capabilities from the thrusters selected. If a propeller is too large for the selected motor, the result will be reduced performance and lower rotations per minute.

For the DIVER ROV, the design team decided to use hobby propellers in the fabrication of the thrusters. Figure 30 shows the Traxxas Model 1583 propellers that were used for the DIVER ROV thrusters (“Traxxas” [70]).

Figure 30: Traxxas Model 1583 Propeller

The higher rake angle on the selected propeller allows for faster acceleration at lower rotations per minute, allowing the DIVER ROV to respond rapidly to changes and reposition. The high rake angle allows the ROV to rapidly accelerate from a stationary position to track a potential target that could be moving in a nonlinear part with non-uniform speed. The last consideration for the DIVER ROV is the material of the propeller. Due to weight and cost restrictions, the team selected the nylon plastic propellers to ensure the best performance to cost ratio. Additionally, the design of the thrusters would allow for these components to be replaced in the future in the event of a component upgrade or a part failure.

3.3.4 Thruster Assembly

The thruster mounting assembly for the ROV was designed under specific considerations. Three thrusters are mounted on the DIVER ROV, providing the propulsion required for a mission. The center thruster is mounted in a vertical orientation and allows for dive and surface motion while the left and right thrusters are mounted about the center axis of the robot and provide forward and reverse motion as well as left and right. Similarly to the design of the DIVER ROV frame, the thruster assembly components were designed for ease of manufacture, ease of assembly, hydrodynamic properties, force and stress concentrations and ease of replacement and upgrade. The center thruster assembly mounts in the center of the ROV in the center thruster mount seen in Figure 31. The center thruster mount can be easily manufactured using the laser cutter and has a distinct keying feature to allow for a quarter rotation to secure the thruster rather than adhesive.

The size frame supports are designed for specific thrusters but can be easily adapted for upgraded thrusters. The triangular design is similar between the front are rear components as well as the holding brackets. The difference between the components is the radius for the thruster. This radius is designed to hold a specific component of the thruster in place and provide support and spacing during operation. The side frame also prevents larger debris from contacting the propeller and becoming entangled. Figures 32 and 33 show the Front and Rear Side Thruster Mounts.

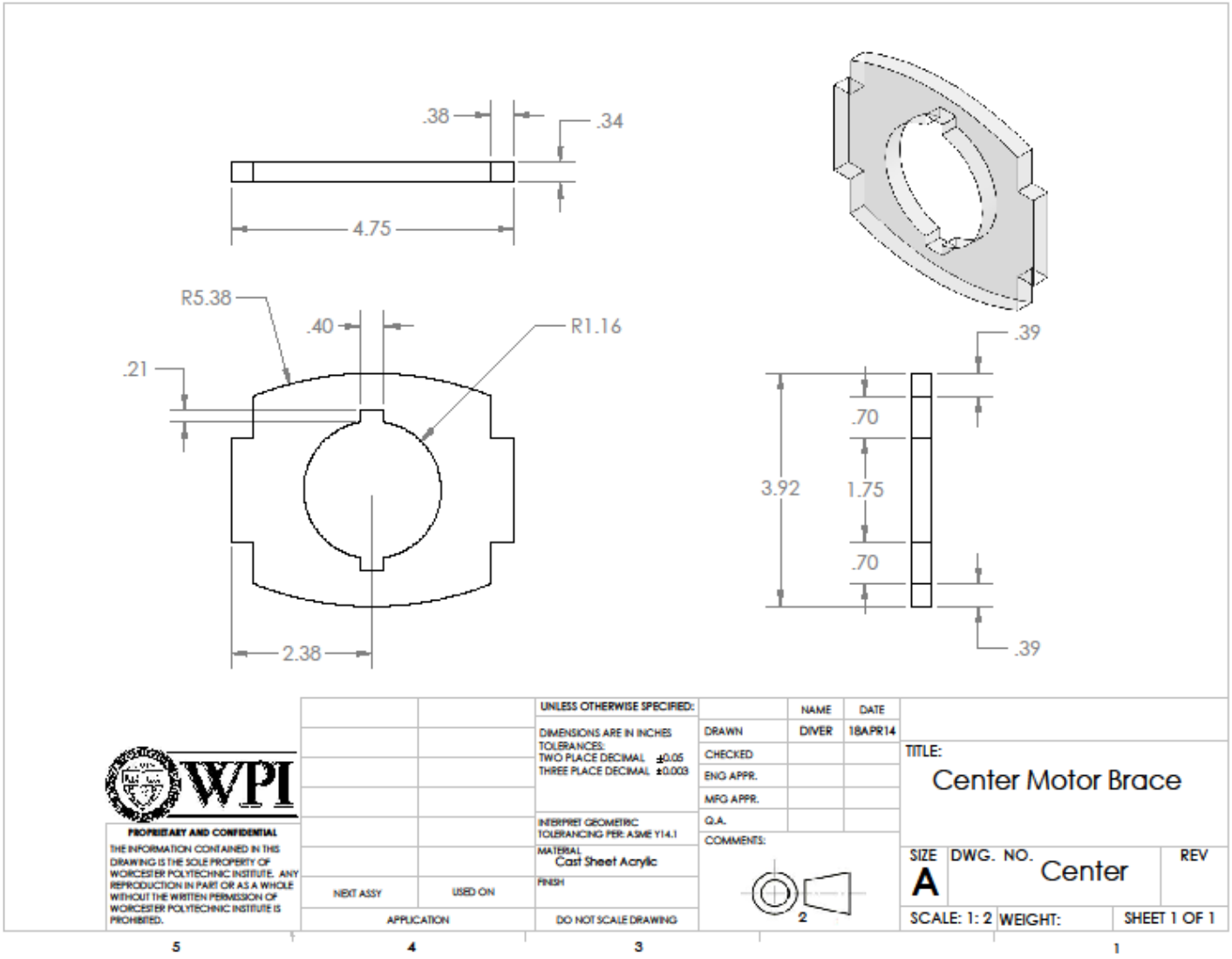


Figure 31: Center Motor Brace

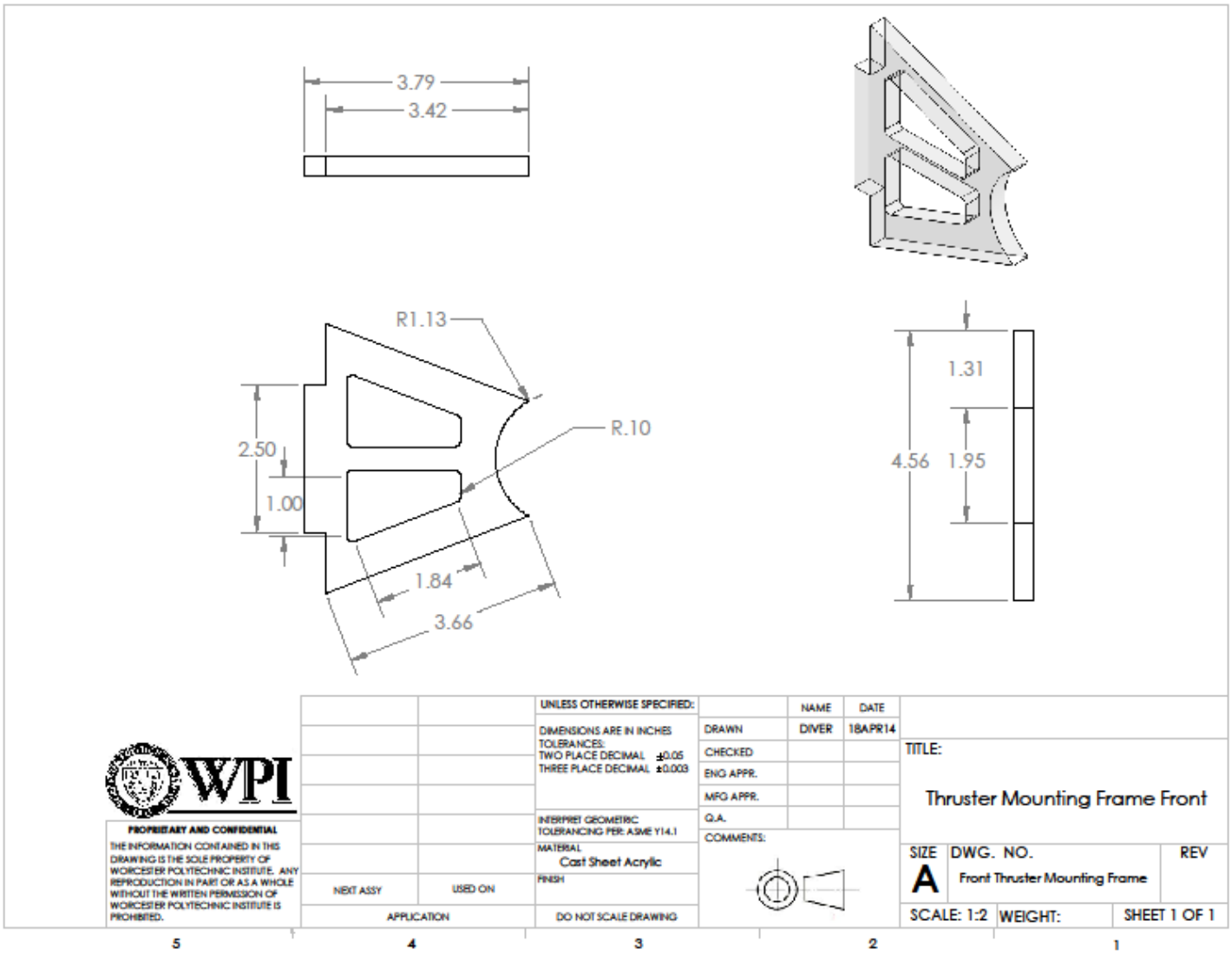


Figure 32: Front Thruster Mounting Frame

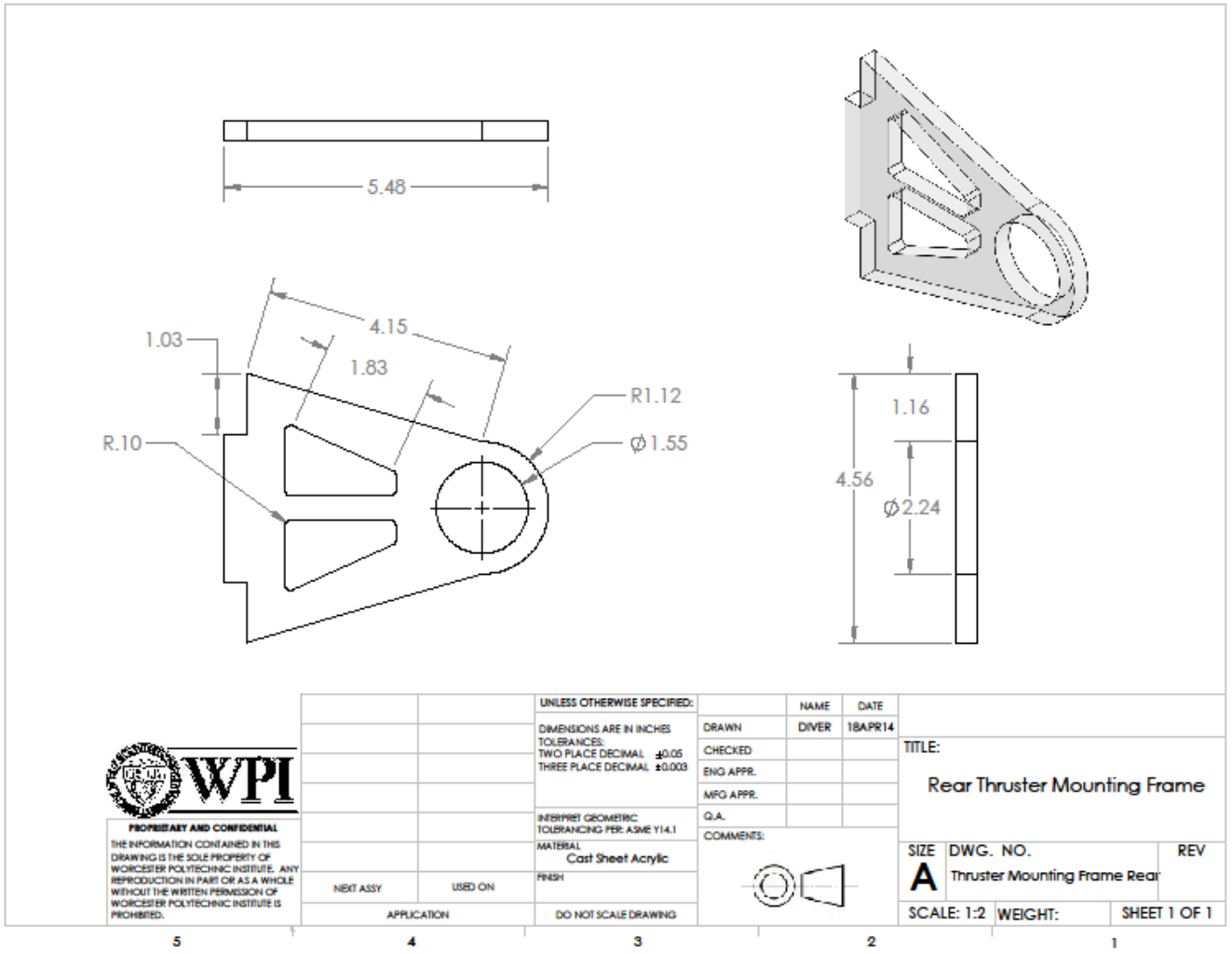


Figure 33: Rear Thruster Mounting Frame

The complete horizontal thruster assembly with the bilge motor, the Traxxas propellers and front and rear mounts can be seen in Figure 34.

Figure 34: Horizontal Thruster Assembly

3.4 Software and Programming

In the following sub-sections, the major programming methods employed for the DIVER ROV are discussed. Programming is a core aspect of any robot, selecting a suitable language for the application can greatly impact the effectiveness of the final product. Two crucial aspects of the DIVER ROV's programming are the framework used to structure and organize the code, Robot Operating System, and the real-time video processing implementation utilized to allow the DIVER ROV to autonomously track a scuba diver. These concepts, as well as how they are implemented, are discussed in detail in sections 3.4.1 and 3.4.2 respectively.

3.4.1 Robot Operating System

Robot Operating System (ROS) is a software framework designed for program development in the robotics industry. Although ROS is not a fully developed operating system, it is a middleware command line interface that provides operating system functionalities such as messaging between processes and package management. The structure of ROS is graph-like; independent executables called nodes send information between one another. This structure allows diverse information to be processed separately and only relevant data must be shared between nodes, to allow the multifaceted nature of robotic programming to be compartmentalized. Because ROS is open source, a suite of user-contributed packages are available for download. This allows users to integrate many tools and functionalities with one another. ROS is used in a number of areas within the industry; most notable is its implementation with the famous PR2 robot shown in Figure 35 ("Robot" [61]).

Figure 35: PR2 Robot

The DIVER ROV project utilizes ROS because it allows many diverse processes to run simultaneously without unwanted interference. Additionally, the wide variety of available packages makes it easy to incorporate many features into the capabilities of the DIVER robot. For example, packages that the team has used to build aspects of the robot's programmable functionalities include a joystick hardware integration package, in addition to a package that allows the team to use an additional processor called an Arduino, which is very well suited for interfacing with hardware and sensors. Because nodes can be written in either C++ or Python and still run seamlessly in tandem, ROS has allowed the team certain liberties and freedom to expand upon

functionalities that would otherwise be incompatible. Because ROS is fairly well known within the industry, many tools and services are ROS-compatible. The team has chosen to take advantage of several of these freely available products, including underwater simulation, and open source video tracking. Figure 36 illustrates the ROS node communication process.

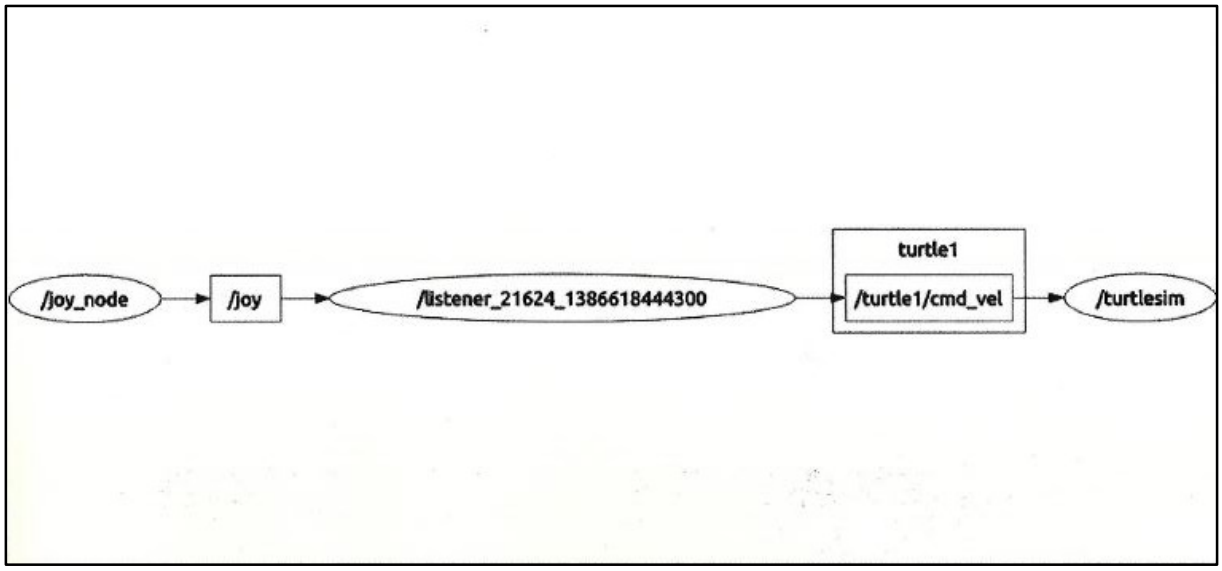


Figure 36: ROS Node Diagram

3.4.2 OpenTLD Video Processing

We begin our overview of the tracking system of the DIVER ROV by defining the local coordinate frame relative to the robot.

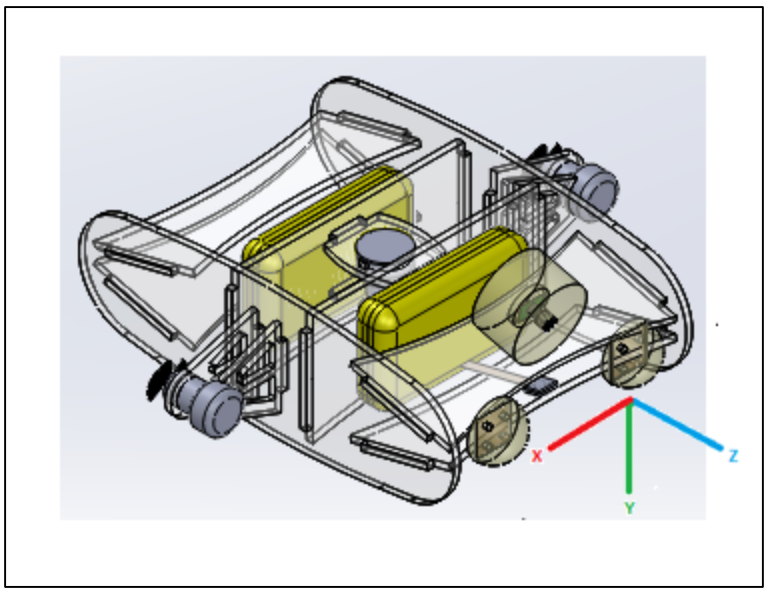


Figure 37: Defining the Local Coordinate System

The video feed's image conveniently maps to the robot's local coordinate system. The X axis is mapped to the horizontal axis in the image, the Y axis to the vertical, and the Z extends through the frame. Note that the selection of the sense of the Y axis is deliberate as it correctly maps objects lower in the frame to a larger magnitude of Y.

In Figure 38 we have a mapping of the video frame captured from the camera located on the front of the robot at any given time. The coordinates are in pixels, with the origin located in the top left of the frame and the center of the frame annotated for clarification.

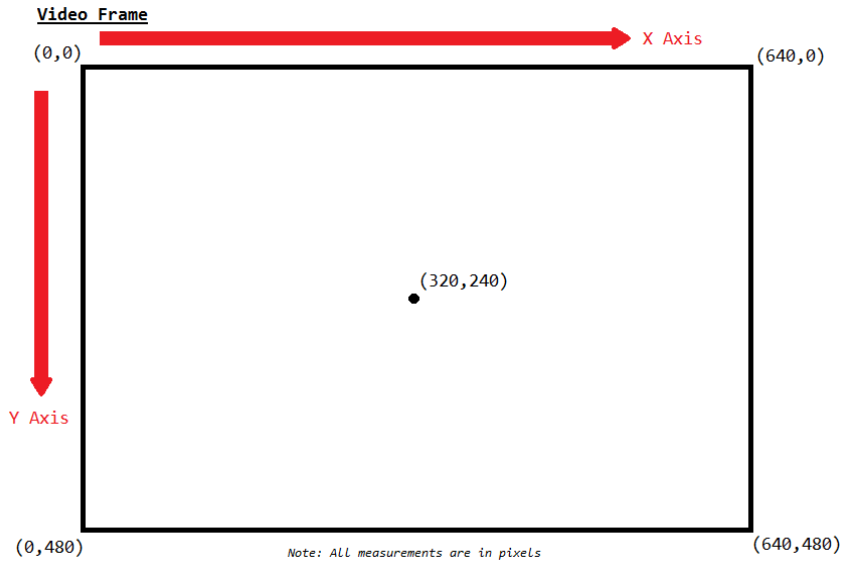


Figure 38: Mapping the Video Frame to the Local Coordinate System

When an object of interest enters the frame (see Figure 39, in this case a scuba diver) the user drags a bounding box about the object to initiate tracking. The size as well as the center point of the box is calculated from the OpenTLD algorithm and passed into the autonomous node.

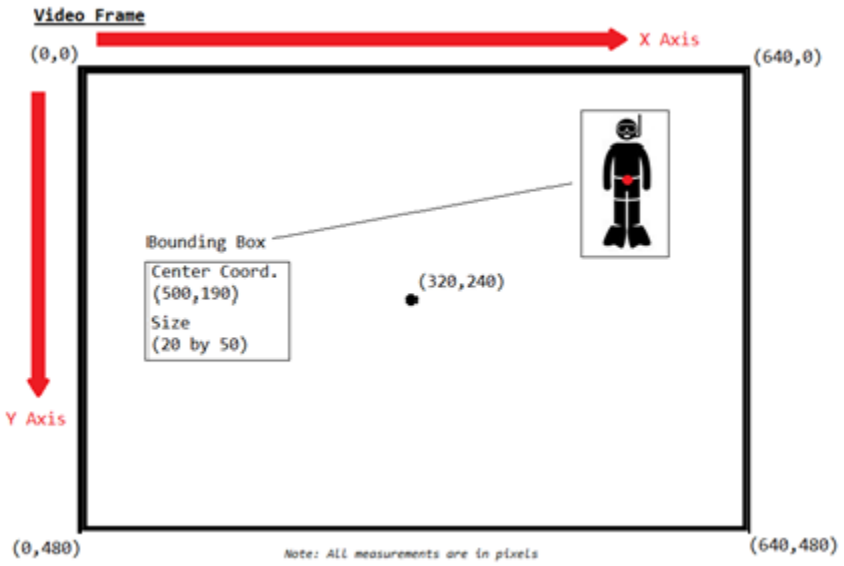


Figure 39: Object of Interest's Bounding Box and Returned Information

The autonomous logic node calculates the offset (Delta) of the current bounding box center coordinates to that of the center of the video frame. Figure 40 depicts the calculation of the offset.

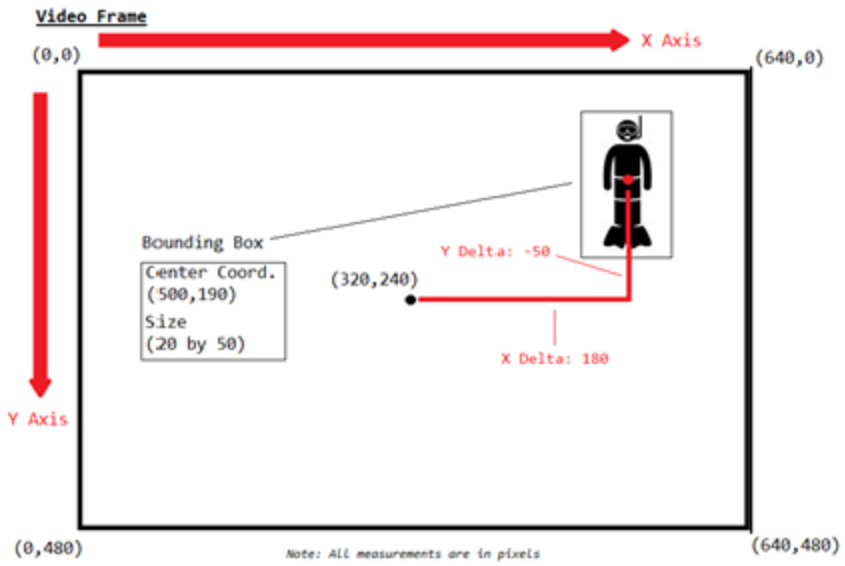


Figure 40: Calculating the Offset from the Center of the Frame

The following calculation is then carried out:

$$Axial\ Thrust\ (\%) = \frac{\Delta\ Coord}{Center\ Coord.}$$

Where Axial Thrust (%) is the percentage of total thrust the system needs to provide relative to its coordinate frame, $\Delta\ Coord$ is the difference between the bounding box center and the center of the video frame on that axis, and $Center\ Coord.$ is the coordinate value of the center of the video frame for that axis. Sign denotes direction. For the X axis, a positive result will produce a turn to the right, while a negative will produce a left turn. Similarly, a positive Y result will produce a change to a deeper depth, and a negative will cause a change to a lesser depth. Note that due to the dampening effect of water, there is no need for a full PID implementation. The system is critically damped and therefore proportional control is acceptable. The thrust values are then passed to the motor control nodes and the motion is executed. In the example featured so far, the resulting thrust would be 56% of maximum in a clockwise direction (about Y, along positive X), and 21% thrust along the vertical plane in the negative Y direction.

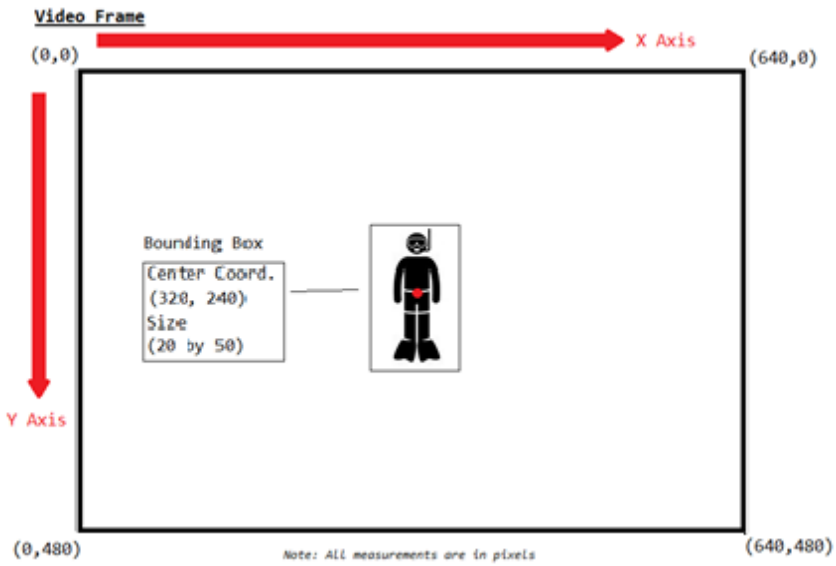


Figure 41: Centered Object of Interest

This process is repeated until the object of interest's center coordinates coincide with the video frame's center coordinates as depicted in Figure 41 (within a 5% tolerance to prevent constant corrections) or until the object leaves the video frame.

In order to maintain distance to the target, a similar method is used. In the following figure, the diver has approached the robot. The resulting diver's image in the frame is larger and the bounding box has a larger area. This is demonstrated in Figure 42.

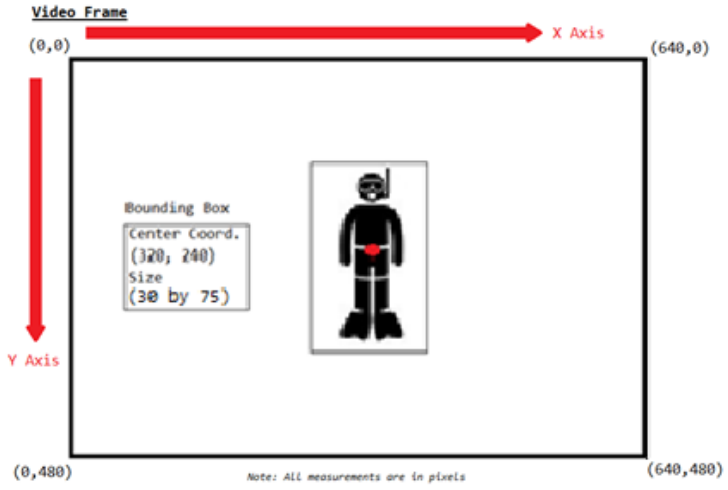


Figure 42: Calculating Changing Difference in Distance to Object of Interest

When tracking was initialized, the bounding box had an area of 1000 square pixels. Now that the diver has approached, the area has increased to 2250 square pixels. The following formula is used to calculate forward/reverse thrust:

$$\frac{Initial\ Area - Current\ Area}{Initial\ Area} = FR\ Thrust\ (\%), \qquad -100\% \leq\ FR\ Thrust\ (\%)\ \leq 100\%$$

Where $FR\ Thrust\ (\%)$ is the amount of thrust desired along the Z axis, $Initial\ Area$ is the area of the bounding box when it was initialized by the user, and $Current\ Area$ is the instantaneous area of the bounding box. Sign denotes direction, with negative sign indicating reverse thrust along the negative Z axis. In the example above, we see the ration of the equation yields -1.25. This value is capped to -1.0 (-100%) and would result in full reverse thrust. As with the previous example, this process is continued until the area falls within a 5% tolerance.

This system is effective in that it allows for any object to be tracked and followed. The generality of its implementation is what gives it versatility. That is, there needs to be no prior knowledge of the object to be followed. However due to this generality there are certain situations that would lead to undesirable behavior. For instance, tracking a long cylindrical object or wide and flat object poses a problem in that the area of the object as viewed through the video frame will rapidly change as its orientation changes. This will result in a non-constant distance to the target as the robot closes or opens distance. This problem is (to our knowledge) unavoidable at the moment given the limitations of a single camera system and our implementation of the OpenTLD algorithm.

3.5 Electronics

Integral to the operation of a robot are its processors, which make computations upon data collected from the surrounding environment. This section discusses the Arduino Mega 2560 R3, and the Raspberry Pi. These two components are keystone elements of the DIVER ROV. In order to collect information about its environment, a robot must have a means of detecting changes within it. This is achieved through the use of sensors. The DIVER ROV contains a suite of such components, and each are thoroughly examined below. Once a robot makes a computation based upon the data collected from the environment, it must enact change in its surroundings. One such way to do this is to move. As such the motor drivers are examined in detail. In order to survive the underwater environment, these electronics must have protection or risk failure. As such, this section describes the techniques used to waterproof the components as well as the selection of a proper housing to contain them in. Assisting commercial divers is part of the DIVER ROV's mission, and our primary means of doing so is through lighting. The steps necessary to create a waterproof light source and implement its use are discussed in this section. As the DIVER robot is also an ROV, it is necessary to have some means of relaying information to a surface operator. The components necessary to do so are a surface station and a tether (for data and power transmission).

3.5.1 Electronic Hardware

The following sections outline the electronic hardware and its technical specifications selected for the DIVER ROV system. This includes the Arduino Mega 2560 R3, the Raspberry Pi, motor drivers, component boxes, and lights. Additional information includes criterion for selection as well as positive and negative attributes for each component. As waterproofing is vital to the survivability of a submersible robot, techniques used to achieve this are covered as well.

3.5.1.1 Arduino

The Arduino Mega 2560 (Mega) is a versatile, open-source microcontroller developed by the Arduino Project. The goal of the project is to make technology more accessible and open the possibilities for creative development (Kushner [39]). The Mega is currently the top of the line microcontroller boasting an AtMega 2560 microprocessor, 8KB of SRAM and 54 I/O pins, as seen in Figure 43 (Kushner [34]).

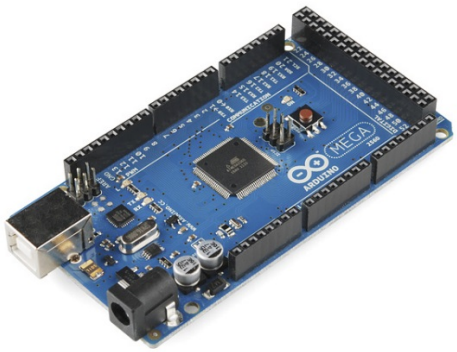


Figure 43: Arduino Mega

It is important to note that currently there are cheaper, less feature rich offerings of the Arduino that could currently serve DIVER ROV's requirements. In the initial stages of design, the DIVER ROV contained additional systems which were never realized in the final design and would have required the Mega's additional functionality. However, it is the team's conclusion that the Mega provides significant flexibility for future expandability in regard to additional sensors and low level peripherals. Therefore, the decision was made to retain the Mega as the choice microcontroller for the DIVER ROV system.

The DIVER ROV integrates several low level sensors, including a dual-purpose accelerometer/gyroscope, electro-magnetic compass, and a force sensitive resistor. The DIVER

ROV also needs to interface with three motor drivers which require low level inputs (specifically, PWM and digital input/output). Therefore, it was necessary to utilize a microcontroller as the Raspberry Pi is incapable of interfacing with this many devices. All told, the DIVER ROV's interface requirements for these devices are 6 digital inputs, 3 digital outputs, 3 PWM outputs, 1 analog input, SDA, SCL and a 5V reference (Vref). The Mega is more than capable of servicing these requirements.

Additionally, the Mega's processor is capable of handling complex computations required for translating raw sensor output to usable data. By pre-processing this data onboard the Mega, the overall processing in the DIVER ROV system becomes distributed between the surface station, the Pi, and the Mega which reduces the possibility of a performance bottleneck. For a complete listing of the specifications of the Arduino Mega 2560 R3, please refer to Table 4.

Table 4: Arduino Mega Specifications

#	**Specification**	**Details**
1	Microcontroller	ATmega2560
2	Operating Voltage	5V
3	Input Voltage (recommended)	7-12V
4	Input Voltage (limits)	6-20V
5	Digital I/O Pins	54 (15 PWM capable)
6	Analog Input Pins	16
7	DC Current per I/O Pin	40 mA
8	DC Current for 3.3V Pin	50 mA
9	Flash Memory	256 KB
10	SRAM	8 KB
11	EEPROM	4 KB
12	Clock Speed	16 MHz

3.5.1.2 Raspberry Pi

The Raspberry Pi is a single board computer originally developed as a means to make an accessible, low price development platform available to students for the purpose of teaching programming. [Raspberry Pi Foundation] Since its inception, it has garnered a following in a much broader spectrum of users because of its capabilities and low price point. The Raspberry Pi was chosen for the DIVER ROV because it is capable of running a full-fledged Linux distribution, has built in Ethernet connectivity, and two USB ports. Linux and Ethernet are requirements for ROS, and USB capability allows for simple communications with the Arduino Mega 2650 R3 the team is using for our low level sensor integration. Although there are other single board Linux capable offerings on the market, all of them at the time of this writing are priced significantly higher than the Pi. Other advantages include a low power requirement, SD card storage, and a small profile. The Raspberry Pi can be seen in Figure 44 (“Raspberry”[55]).

Figure 44: Raspberry Pi

The DIVER ROV is running the Raspbian Linux distribution. Raspbian is based on the Debian family of Linux operating systems and comes pre-configured for some of the hardware differences between a Pi and a standard desktop PC. A concrete example of a difference is the existence of the GPIO pins on the Pi. Raspbian makes them available through software without needing special configuration, which would not be true of a standard Linux distribution. The DIVER ROV's install of Raspbian is stock with no major changes required for usage with the robot. One minor exception to the stock setup is more of a convenience issue than anything else; The DIVER ROV's install of Raspbian includes a custom written startup script that will execute ROS Hydro and the required nodes for operation. Without this script in place, the operator would be required to SSH into the Pi, and execute the nodes manually.

The Pi is being powered via the DIVER ROV's onboard 5V power supply bus provided by a switching regulation. This is accomplished through a simple breakout board provided by Sparkfun Electronics (Sparkfun part number BOB-10031). The decision to use a breakout board and power the Pi through the USB circuitry rather than use the exposed 5V power pin is due to the fact that the USB circuitry is regulated and the exposed pin is not. Although protections are in place to minimize the amount of fluctuation on the DIVER ROV's 5V power bus, there is no guarantee that the bus will be able to provide a constant 5V. Given that the Pi, like all computers, is susceptible to data corruption or failure in a brown-out or over-voltage scenario, using the regulated circuitry provides an additional layer of protection against such an occurrence. For a list of specifications of the Pi, please refer to Table 5.

Table 5: Raspberry Pi Specifications

#	Specification	Details
1	Developer	Raspberry Pi Foundation
2	Type	Single-board computer
3	Price	$35
4	Operating systems	Linux, RISC OS, FreeBSD, NetBSD, Plan 9
5	Power	3.5 W (model B)
6	CPU	ARM1176JZF-S (ARMv6k) 700 MHz
7	RAM	512 MB
8	Storage	SD Card (Min. 4GB)
9	Graphics	Broadcom VideoCore IV

3.5.1.3 Motor Drivers

A motor driver is an electronic device which acts intermediately between a motor and a microprocessor and power source. Because motors require higher voltage and current than a microprocessor alone can provide, the two must work together to power and control the motors. For this project, Pololu 15 Amp motor drivers seen in Figure 45 were selected. Because of budgetary restrictions, the motor drivers were selected much earlier in the design process than the motors. Therefore, it was necessary to select powerful motor driver which would be compatible with a wide range of options. The Pololu motor drivers selected can support a nominal motor voltage between 5.5 V and 24 V, a very wide range. Additionally, these motor drivers can deliver up to 15 A continually. This high range provided the team with versatility and options when selecting motors. A Pololu motor driver can be seen in Figure 45 (“Motor” [45]).

Figure 45: Motor Controller

The Pololu motor drivers operate by receiving a pulse width modulation (PWM) signal from the Arduino, and using this to modulate the control of the motors. The motor drivers can be implemented one of two ways, sign-magnitude or locked-antiphase. Sign-magnitude allows the speed of the motors to be controlled by the PWM signal, and the direction of the motors to be controlled by a separate direction pin, which is set high or low by the Arduino. Locked-antiphase changes the direction of the motor based on ranges of PWM signal, and the direction pin is not considered. For simplicity in computing, the team has chosen to implement the sign-magnitude method. Because the robot has three motors, one on each side, and one centrally, three Pololu 15 Amp motor drivers have been implemented. The motor drivers produce heat more rapidly than the other electronics, so they are being housed in one of the two dry boxes with the Arduino, separately from the sensors, since proximity to the motor drivers can interfere with some sensor readings. The technical specifications can be seen in Table 6.

Table 6: Motor Driver Specifications

#	**Specification**	**Details**
1	Operating Voltage	5.5 V to 24 V
2	Current Output	15 Amps
3	Peak Output Current	170 Amps
4	Max PWM Frequency	40 kHz

3.5.1.4 Component Box

The electronics housing for the DIVER ROV consists of two commercially available waterproof cases produced by OtterBox. The 3000 series OtterBox Drybox case is used for the mounting location for all of the major electronics such as the Arduino Mega and the Raspberry Pi. To protect these electronics throughout the mission, the team selected the OtterBox Drybox 3000 series. The 3000 series is fully waterproof up to 100m, allowing the DIVER ROV to be fully functional within its required depth range. The 3000 series OtterBox Drybox is small enough to easily fit onboard the ROV while fully containing all of the electrical components. Figure 46 depicts the OtterBox Drybox 3000 and Table 7 lists its specifications (OtterBox [51]).

Figure 46: OtterBox Drybox 3000 Series

Table 7: OtterBox Drybox 3000 Specifications

#	Specification	Details
1	Manufacturer	OtterBox
2	Model	Drybox 3000
3	Material	Polycarbonate
4	Exterior Dimensions	8.813" x 5.175" x 2.008"
5	Interior Dimensions	7.639" x 3.723" x 1.229"
6	Interior Volume	35 Cubic Inches
7	Waterproof Rating	100 Feet

3.5.1.5 Waterproofing

Given the aquatic environment that the DIVER ROV operates in, waterproofing the electron components was an essential task. Two chemical products were used to waterproof various components of the DIVER ROV. The first product is 3M 5200 Marine Adhesive Sealant. This sealant is polyurethane based and provides a “watertight” and “weather resistant” seal, according to the product label description. The 5200 sealant came in a tube and required a week long curing time. The second chemical product used to waterproof the DIVER ROV was PARKS Super Glaze Ultra Gloss Epoxy. This product consists of a liquid resin and activator, which cure to form a solid when combined in equal parts.

3.5.1.6 Lights

Because sunlight diffuses rapidly underwater, an additional light source is an essential part of any ROV. Providing a light source underwater is a difficult task because of how quickly light is absorbed under water. Water contains more suspended particles than air, such as microscopic organisms, debris and plant matter, which can cause light to bounce and reflect. This effect, shown in Figure 47, is known as backscatter (Read [60]).

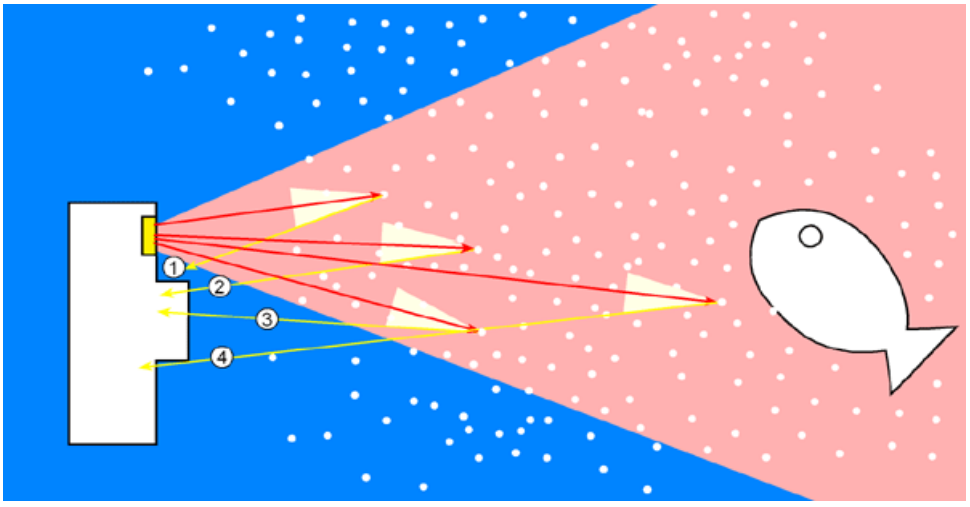


Figure 47: Light Underwater

Backscatter causes underwater lighting to be much less effective. In order combat the dimming due to backscatter, the team has chosen to implement two LED arrays, one on either side of the camera. The circuit used for each array can be seen in Figure 48, where the 5 volt source voltage is powered from the Arduino.

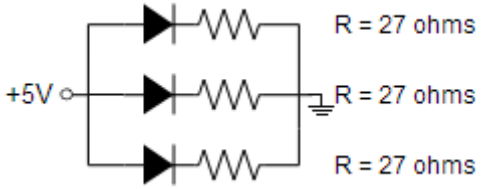


Figure 48: Lighting Circuit

By using LED arrays on either side, rather than single LEDs, the directional effect of the light dispersion is minimized. This allows a greater lighted workspace for the robot's trajectory. Another way in which the team has maximized the robot's light source is by angling the lights inward as shown in Figure 49.

Figure 49: LED Light Arrays on Front of DIVER ROV

This concentrates the effect of the lights in an area centered in the camera's direct field of vision, ensuring that the most crucial area is illuminated for maximum visibility. Positioning the lights away from the camera lens reduces the effect of backscatter by distancing the source of reflection from the video capturing area. The LEDs utilized for this project are 5mm LEDs with a 50 degree viewing angle illustrated in Figure 50 ("China" [12]).

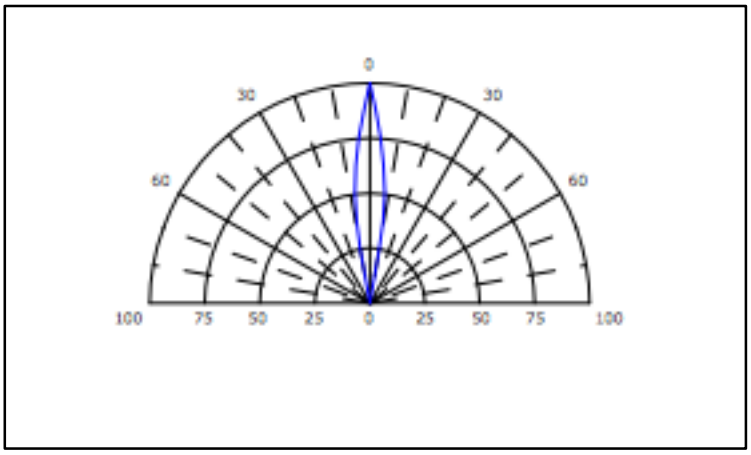


Figure 50: LED Viewing Angle Illustration

Because this is a very narrow viewing angle, some enhancements had to be made in order to decrease the strict directionality of the lights. The LED arrays were made waterproof by submerging them in clear epoxy. When the epoxy had set, the surface was sanded and filed in order to make it cloudy and therefore diffuse the light further. The team estimates that this increased the viewing angle by about ten percent.

The assistive aspect of the lighting system is based on the utility of a constant light source following a diver underwater. Additionally, the surface operator is able to communicate with the diver by sending a message through the graphical user interface. This message is then outputted to the LEDs using Morse code. In this way, the surface and the diver can interact. Although Morse code is cumbersome and obscure, implementing it successfully in this way opens up the possibility

of creating a series of preset commands and messages which the diver can learn ahead of time without having to be strictly fluent in Morse code. This effect is implemented using ROS's message passing feature, where the graphical user interface and the Morse code implementation exist on separate nodes. The graphical user interface posts the user input to a topic, which is then subscribed to by the Morse code node. The Morse code node uses a Morse code Table to translate each letter, and then set's the LED pulse width modulation pin high and low for the proper time intervals to produce long and short flashes. Thus, the surface station can communicate quickly and safely with the diver.

3.5.2 Sensors

In the following sub-sections, the sensors integrated for the DIVER ROV are discussed. Sensors are a key part of any robot because they allow the system to interpret and respond to its environment. Selecting the right combination of sensors ensures that the robot has all the information it needs to react intelligently to its surroundings. The sensors selected for the DIVER ROV are a camera, pressure sensor, electronic compass and gyroscope. Their functionality and implementation are discussed in the following sub-sections.

3.5.2.1 Camera

A high quality, high speed camera is an important aspect of observation class ROVs, as it allows the operator to see into the robot's environment. For the DIVER ROV project, the camera is an indispensable component because it provides an outlet for the video processing software, OpenTLD. For this application, the PlayStation Eye was chosen. The PlayStation Eye was originally designed for use with the PlayStation gaming console as part of a live interaction feature. This makes the camera ideal for this application because it was specifically designed for use in low lights with real time video processing in mind. The camera features a field of view of 75 degrees, as seen in Figure 51, and a frame rate of 60 frames per second with a resolution of 640 x

480 (“PlayStation” [54]). The PlayStation Eye is a USB camera, meaning it can be easily interfaced with the Raspberry Pi.

Figure 51: Sony PlayStation Eye

The Specifications of the camera can be seen in Table 9.

Table 8: Sony Eye Specifications

#	Specification	Details
1	Resolution	640 x 480
2	Frame Rate	60 frames/second
3	Connection	USB 2.0
4	Dimensions	3.1 x 6.8 x 8.9 in
5	Weight	9.6 ounces

The fabrication of the DIVER ROV required the camera to be fully waterproofed prior to integration onto the robot. To achieve this, the camera circuit board was removed from the plastic housing and stand. A 2 inch by 2 inch piece of 1/8 inch glass was cleaned and placed on the counter top. The camera lens was then sealed to the glass piece using superglue, as seen in Figure 52. The purpose of sealing the camera lens to the glass was to prevent any epoxy from getting into the small gap between the glass and camera lens, which would render the camera useless.

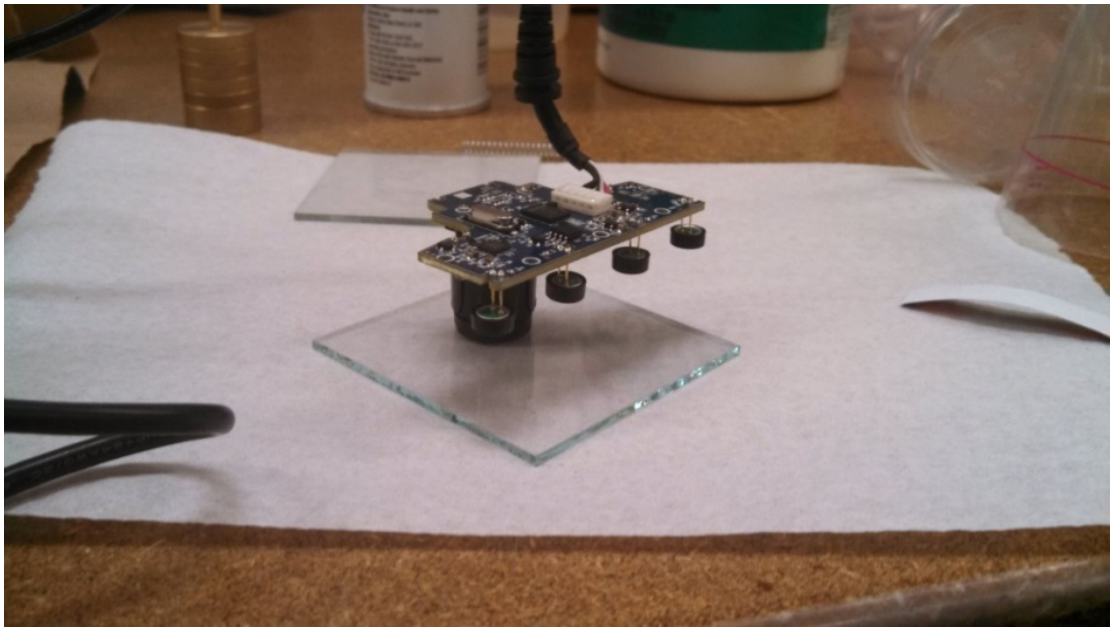


Figure 52: Camera Waterproofing System

The next step was to waterproof the circuit board. To accomplish this, a two part urethane epoxy was used. The glass-camera assembly was placed in a mold and carefully positioned to ensure there was no space on the bottom of the glass. For aesthetics, the DIVER ROV logo was added. The epoxy was mixed and added to the mold as seen in Figure 53.

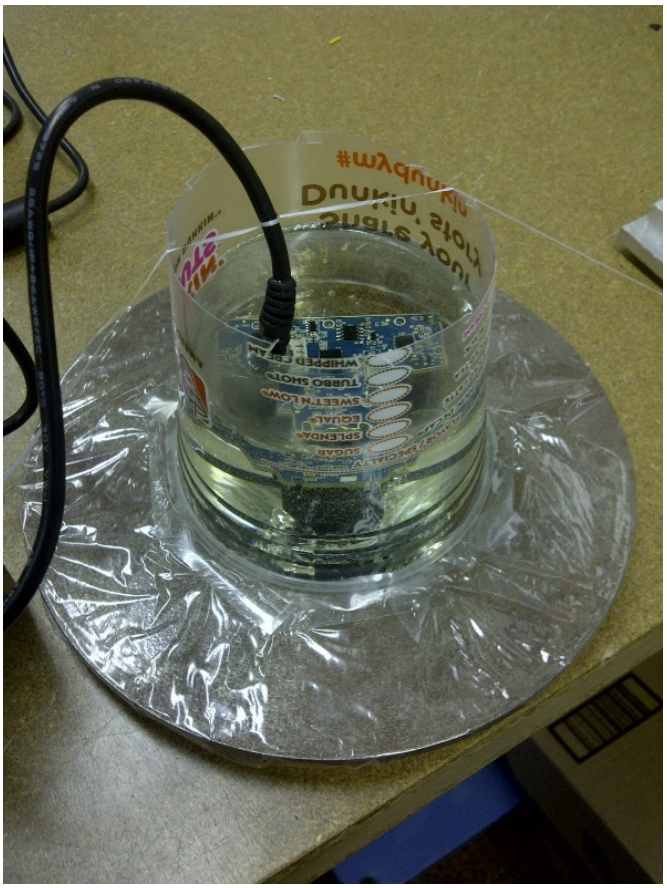


Figure 53: Epoxy Application to Waterproof Housing

After the epoxy fully cured, the camera housing was removed from the mold. The result was a solid block of epoxy with the camera circuit board sealed inside. While the mold was larger than the camera, having extra epoxy around the edges allows for room to alter the housing when mounting and not threaten unsealing the circuit board. The final camera assembly can be seen in Figure 54.

Figure 54: Camera Housing Completed

The last step in the fabrication of the camera assembly was to ensure it was fully functional. The camera was hooked up to the lab computer and tested. The result was a fully functional and extremely clear video, as seen in Figure 55.

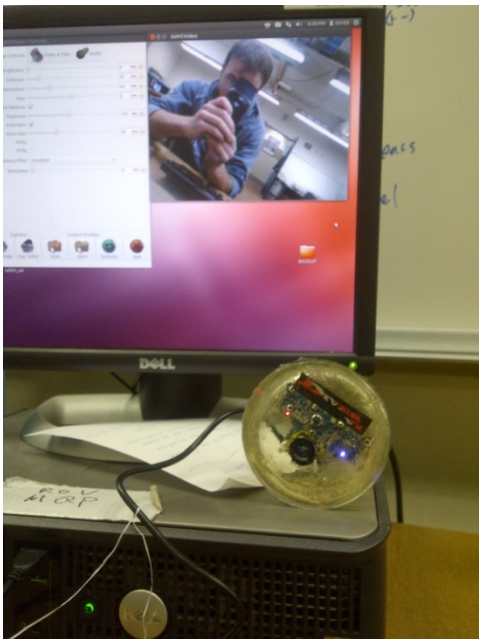


Figure 55: Camera Module Testing

3.5.2.2 Pressure Sensor

The depth of the ROV is important information that allows the robot avoid crush depth and ensure its safety. In order to determine the depth, a pressure sensor has been implemented to read the atmospheric pressure of the robot's environment, which is in turn used to extrapolate the depth using Arduino code. The output of the sensor is based on the pressure applied to it, and the static pressure on the sensor can be used to determine the depth of the robot based on the equation,

$$P_{static} = \rho g h$$

where ρ represents the density of water, g represents the acceleration of gravity, and h represents the depth of the robot. Since the density of water and the acceleration due to gravity are both known values, and the static pressure is determined by the sensor, the depth of the water is the only unknown when this equation is applied. Therefore, a single pressure sensor is all that is needed to determine the depth of the robot with a reliable degree of accuracy.

The depth sensor chosen for the DIVER ROV project is a Flexiforce 25 pound pressure sensor seen in Figure 56 ("FlexiForce" [23]).

Figure 56: Parallax Pressure Sensor

The technical specifications of the sensor can be seen in Table 10.

Table 9: Parallax Pressure Sensor Specifications

#	Specification	Details
1	Load Capacity	25 lbs.
2	Accuracy	±3%
3	Output Type	Analog
4	Sensing Area	0.11 in^2
5	Response Time	<5 microseconds

To measure pressure, the analog sensor uses two ultra-thin conductive surfaces which decrease in resistance when pressure is applied to them. The sensor can be connected with a very simple circuit and a single resistor.

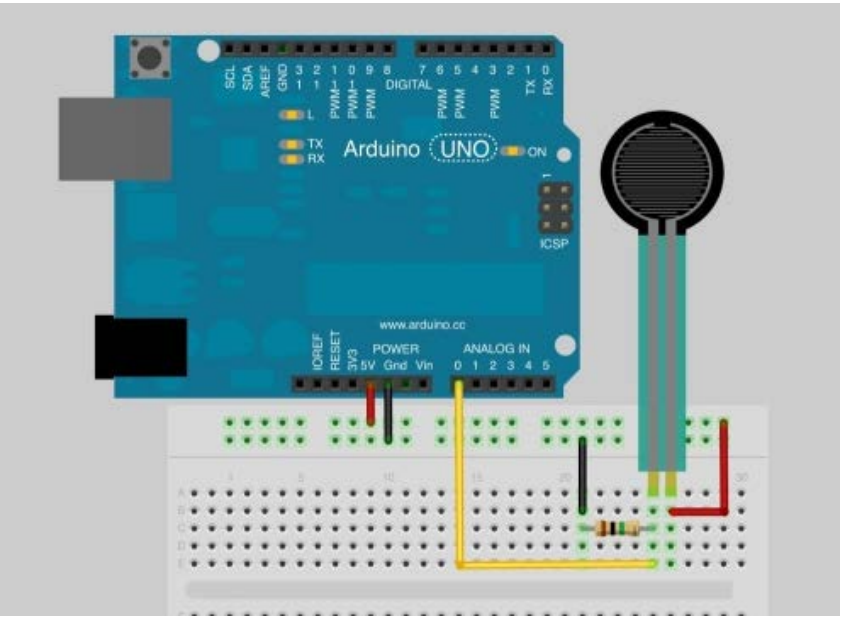


Figure 57: Wiring Diagram for Pressure Sensor

Figure 57 shows a simplified example of the connection of the pressure sensor and the Arduino (“Flexiforce Pressure” [22]). The values that the sensor outputs range from 0 to 1023, but in order to give meaning to these values, the sensor must first be calibrated. Calibrating the

pressure sensor is done by setting it up to print sensor values serially while applying known weights and recording data points. The results of the sensor calibration can be seen in the graph below.

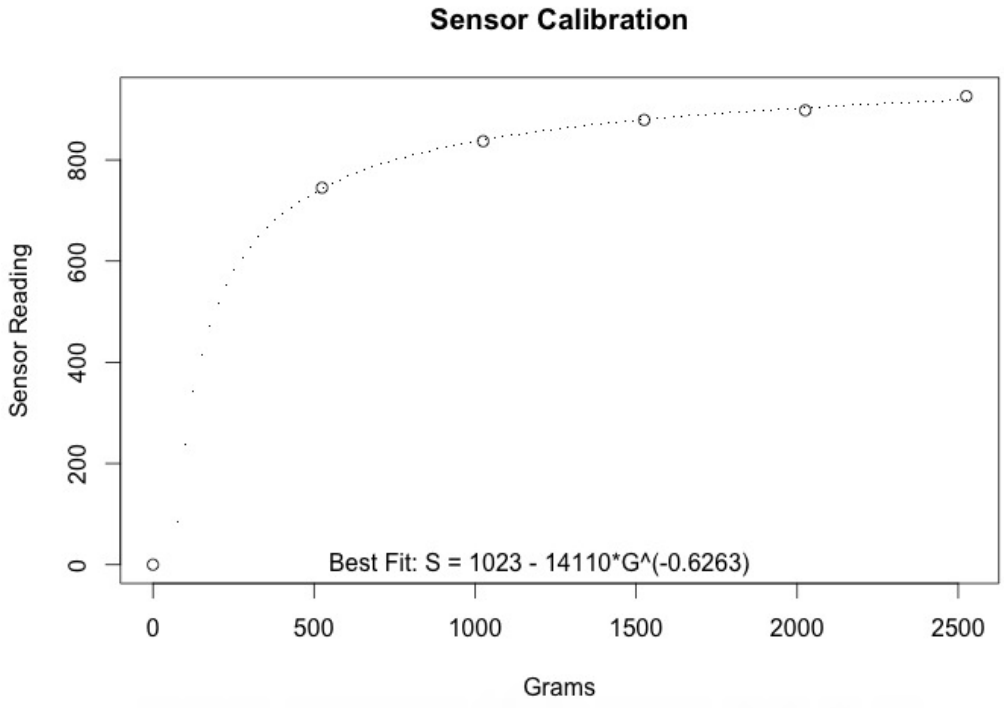


Figure 58: Pressure Sensor Calibration Data

This graph displays the relationship between the weight applied to the sensor in grams, and the outputted sensor value. By applying a best fit line to these points, the estimated weight on the sensor can be extrapolated for any sensor value. In order to limit the processing load on the Arduino, the complex function used to relate sensor output to weight is broken up into ten linear functions which can be referenced quickly by the processor to access the estimated weight applied to the sensor. Ultimately, the weight in grams is converted into pounds of pressure and finally to feet below sea level. Since the area of the sensing surface and the PSI per foot of depth are known, the pressure read by the sensor can be directly linked to the depth. Once the depth is determined, this data is posted to a topic using ROS. The ROS topic makes the depth data available to the graphical user interface, allowing the surface operator to know the depth of the ROV in real time.

3.5.2.3 Electronic Compass

The orientation of an ROV can be difficult for the surface operator to perceive based on visual cues alone. Implementing an electronic compass provides the surface operator with the heading of the robot which is vital to navigate the robot. The compass selected for this project, seen in Figure 59, is a LSM303 tilt compensated compass, meaning that it combines a triple axis magnetic sensor with a triple axis accelerometer (LSM303DLMTR [37]). This allows the compass to maintain accuracy of the robot's heading regardless of whether the robot remains level. Although land bound robots typically operate in a two dimensional plane on the ground, this high degree of accuracy is necessary for robots operating in three dimensions such as quadcopters and ROVs.

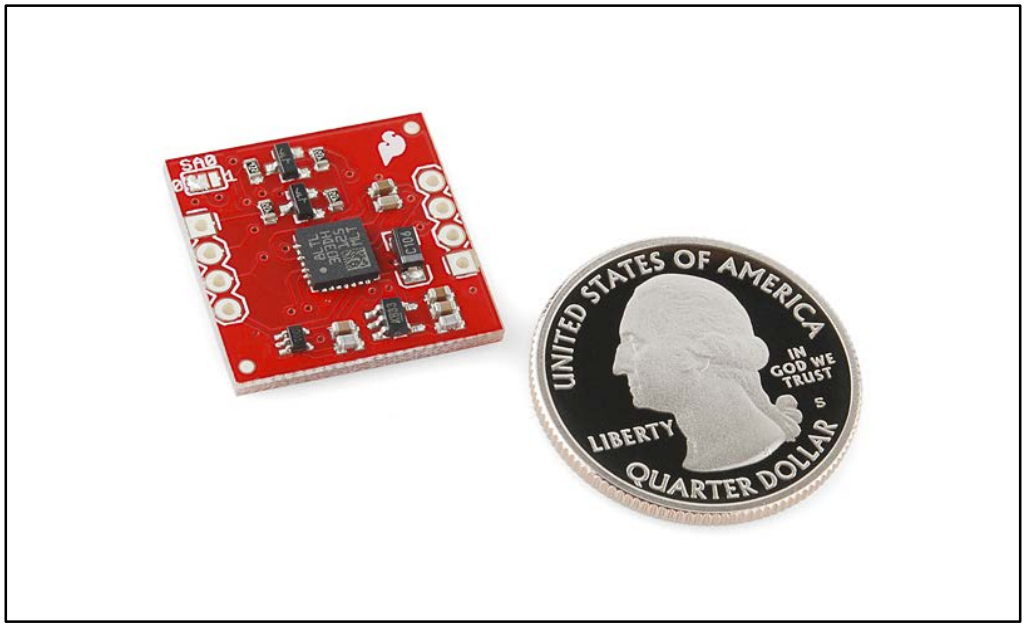


Figure 59: LSM303 Tilt Compass

The LSM303 was chosen for its good availability, small size, and high accuracy. The technical specifications of the compass can be found in Table 11.

Table 10: LSM303 Compass Specifications

#	Specification	Details
1	Supply Voltage	2.16 V to 3.6 V
2	Connection	I^2C Serial
3	Output Type	16-bit data out
4	Magnetic Field Scale	+-1.3 to +- 8.1 Gauss

The compass communicates serially with the Arduino Mega, which receives and interprets the sensor data. From the Arduino, the heading of the ROV is posted to a topic using ROS. Through ROS's message passing capabilities, this topic makes the robot's heading data available to the graphical user interface, allowing the surface operator to understand the orientation of the robot in real time. The electronic compass is housed in one of the two dry boxes being used to house the sensors, Raspberry Pi, Arduino, and motor drivers.

3.5.2.4 Gyroscope

Because of the difficulty of remotely operating a robot through a video feed alone, it is important to incorporate additional sensing data pertaining to the orientation and attitude of the DIVER ROV. Having knowledge of the robot's yaw, pitch, and roll enables the surface operator to make intelligent choices about the navigation of the robot. Figure 60 depicts pitch, roll and yaw ("Photobucket" [53]).

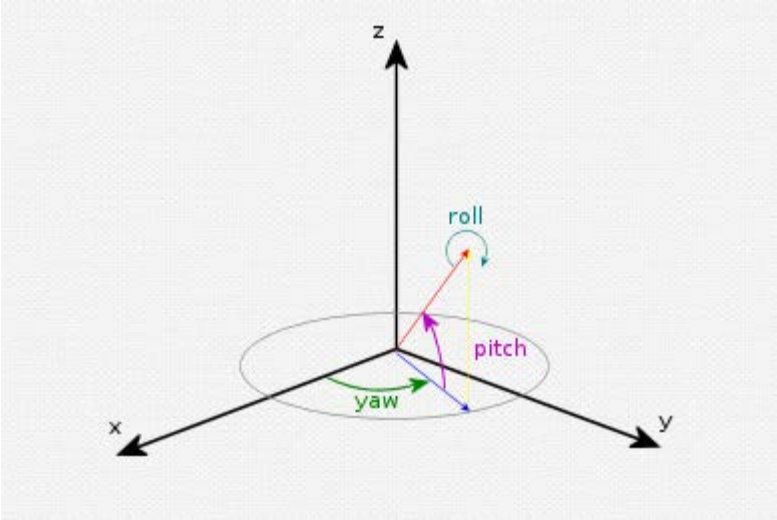


Figure 60: Pitch Roll Yaw Diagram

For this project, this information is captured by implementing a gyroscopic inertial measurement unit (IMU) called the Kootek® Arduino GY-521 MPU-6050 Module 3 axial gyroscope accelerometer stance tilt module. This is a generic sensor adaptation of a common microprocessor unit (MPU), MPU-6050. This selection was made because of its low cost, high availability, and ample documentation. The IMU communicated serially through the Arduino, which posts yaw, pitch and roll values to a ROS topic. In turn, the graphical user interface accesses this topic and displays this valuable information to the surface operator. The technical specifications of the MPU-650 can be seen in Table 12.

Table 11: MPU 650 Specifications

#	Specification	Details
1	Supply Voltage	3 V to 5 V
2	Connection	I^2C Serial
3	Output Type	16-bit data out
4	Weight	0.5 ounces
5	Dimensions	2.4 x 1.7 x 1 inches

3.5.3 Surface Station

The surface station is a Sony Vaio VPC115FM laptop computer, shown in Figure 61, running the Ubuntu 12.0.4 "Precise Pangolin" operating system (“Sony” [63]). This computer was chosen due to its high system specifications in order to produce the highest possible performance with the OpenTLD algorithm. Tests run by the team have shown that an average of 62 frames per second are attained when tracking an object with a free and uncluttered background. Prior to choosing this platform, the same test was run with identical conditions on an Asus EEE PC Netbook with a 1.0GHz Intel Atom Processor and 1GB of RAM which only garnered an average of 9 frames per second.

Figure 61: Sony Vaio VPC115FM Laptop Computer

While the processing power of the Vaio is significantly higher than that of the EEE PC, that power comes with a significant drawback. The operating time of the Vaio's battery while running the algorithm consistently came in at 116 minutes, while the EEE PC was able to run the algorithm for six hours. This drawback can be mitigated by carrying spare batteries, but while the

battery change operation is taking place, the DIVER ROV is incapable of operating. Therefore, a direct power source of 110V AC is desirable when undertaking lengthy missions. This can be provided from a wall outlet, a DC-AC inverter, a gasoline generator, or an emergency battery backup pack.

The Vaio also has a 500GB hard drive, which makes video capture feasible without an external storage medium. This is highly desirable for certain applications, such as marine research and inspection missions, where review of the footage at a later time can be critically important. The Vaio also has three USB 2.0 ports as well as an e-SATA port making the use of external drives feasible for video capture as well. Other features that are desirable are a backlit keyboard (for night usage), built in gigabit network interface card for high speed communications, and a DVD burner for creating optical backups of data. There are a number of ROS nodes that must be run on the surface station. Specifically, these nodes are responsible for:

- Joystick/Gamepad input
- Graphical User Interface
- OpenTLD

The entirety of the rest of the DIVER ROV program is run on the Raspberry Pi. The list of relevant system specifications can be found in Table 13

Table 12: Surface Station Specifications

#	Specification	Details
1	Processor Make	Intel
2	CPU Family	Core i7
3	CPU Model	I7-720QM
4	CPU Speed	1.6GHz
5	Cache Size	6MB L3 cache
6	System RAM	6GB
7	GPU	NVIDIA GT 330M
8	Video RAM	1GB Dedicated
9	Battery Type	Lithium Ion
10	Battery Life	Approx. 2 Hours*
11	Battery Access	Removable

3.5.4 Power and Tether

As discussed in earlier sections, a ROV can be controlled wirelessly or using a tether. The DIVER ROV team selected a tethered ROV system because it allows for uninterrupted data and power transmission. The tether system for the DIVER ROV consists of a direct current power line and a shielded CAT5e Ethernet cable. Both cables are run from the surface power station directly into the waterproof housing onboard the ROV. The tether system links the network interface controllers on the Raspberry Pi and the surface station computer. This process is known as crosslinking, allowing the network interface controllers to directly communicate without the use of a router. Figure 62 shows the tether of the DIVER ROV.

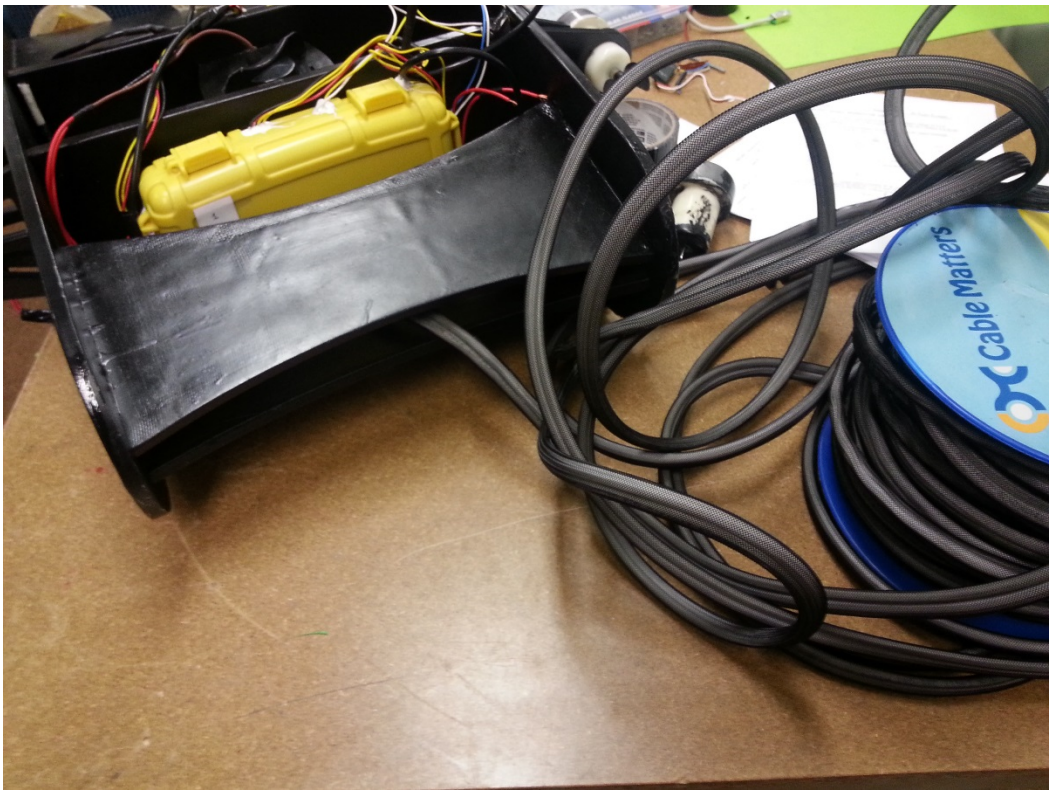


Figure 62: DIVER ROV with Tether

3.6 The Complete DIVER ROV

Figure 63 show the completed prototype for the DIVER ROV. As stated earlier, the frame of the ROV consists of fiberglass reinforced cast acrylic, machined using a laser cutter. Three bilge pump thrusters provide mobility while two OtterBox Drybox 3000 waterproof cases contain the onboard electronics. A front mounted camera and a light array set provide video feedback via tether to the surface operator at the surface station. The final ROV occupies a total volume of 398 cubic inches and weighs 17.2 pounds, as seen in the SolidWorks Drawing in Figure 64. The system is able to be deployed in under ten minutes and can remain operational over one hour.

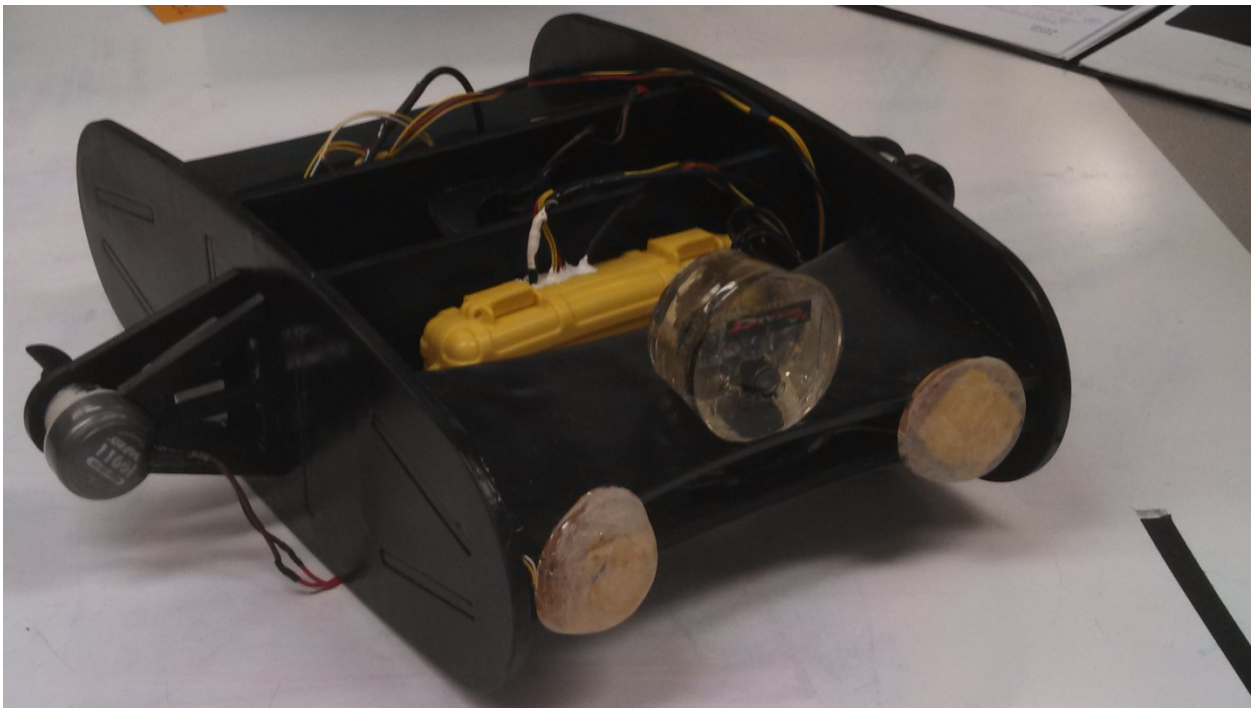


Figure 63: Completed DIVER ROV

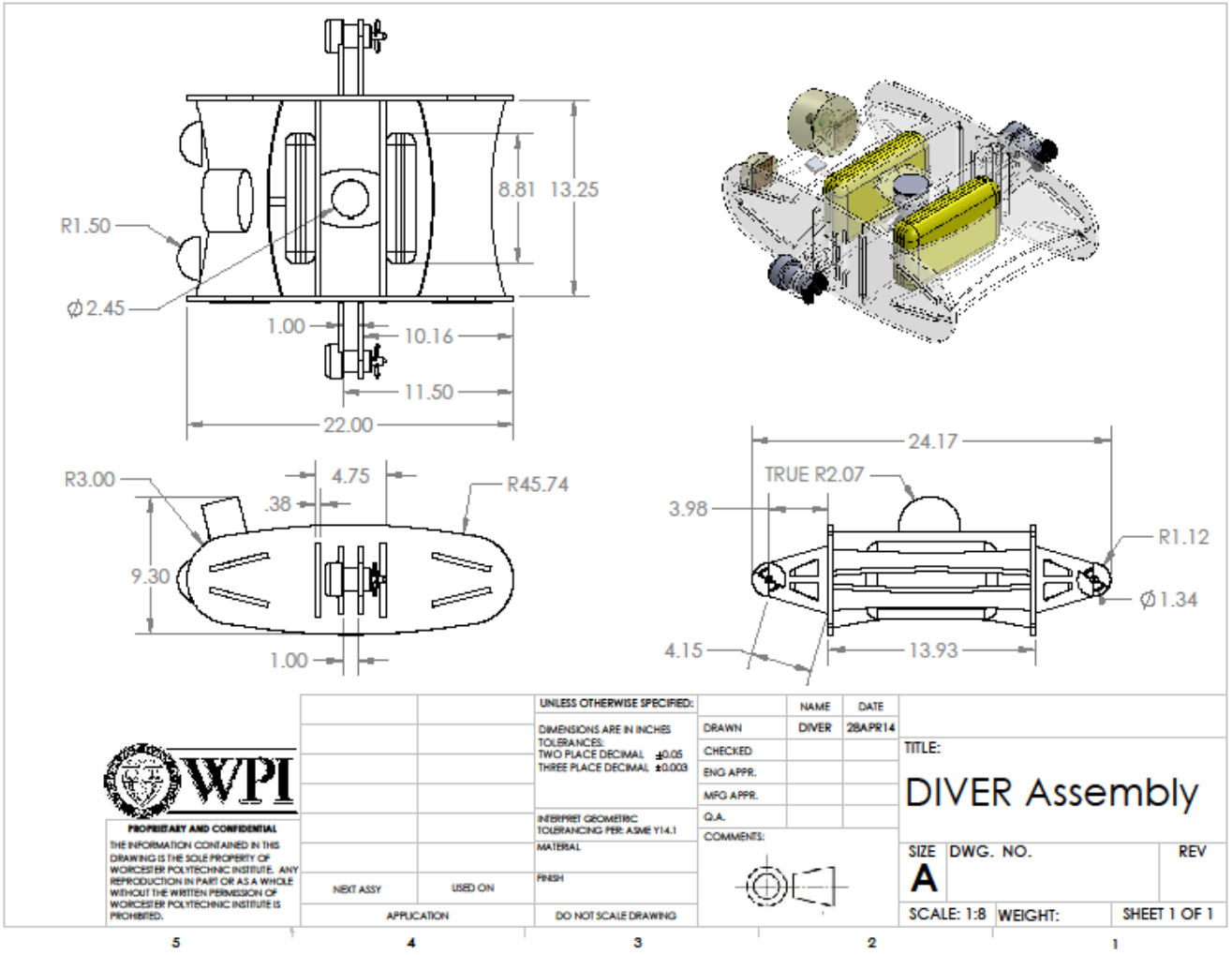


Figure 64: DIVER ROV Assembly Drawing

3.7 Controls Overview

Control of the DIVER ROV system consists of ROS nodes operating in three primary locations. These locations are the surface station, the Raspberry Pi, and the Arduino Mega as seen in Figure 65.

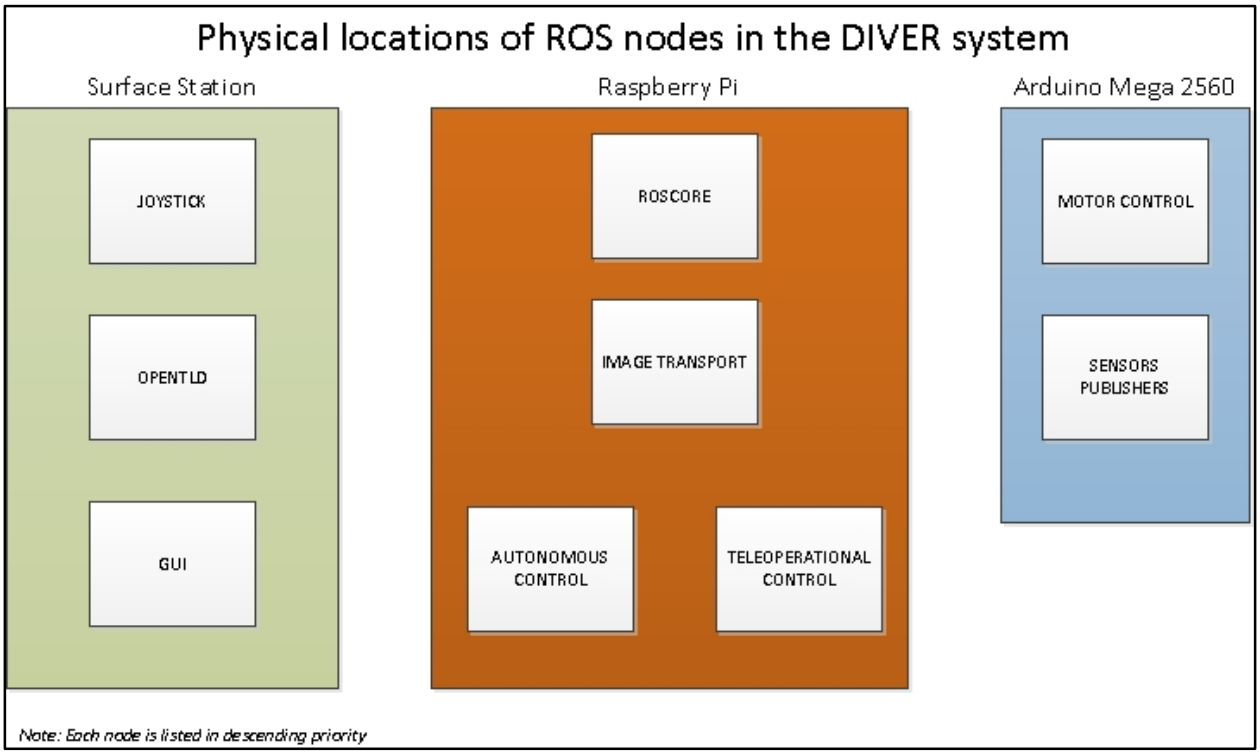


Figure 65: Physical Locations of ROS nodes in the DIVER ROV System

On the surface station, there are three nodes in operation. These are the Joystick node, the OpenTLD node, and the Graphical User Interface (GUI) node. The Joystick node is run here because it requires direct access to the physical joystick (the Xbox 360 gamepad) which is connected via USB to the computer. OpenTLD is run here to take advantage of the surface station's significant processing power so that the image processing may occur at the highest frame rate possible. Latency in the video processing leads to errors in tracking, which significantly impacts the autonomous capability of the robot. Finally, the GUI is run here due to the need for direct input from the mouse and keyboard.

The Raspberry Pi runs three nodes at any given time. These are ROSCore, Image Transport node, and either the Autonomous Control Node or the Teleoperational Control node (depending on which mode the robot is currently running). ROSCore is essential to the ROS framework and acts as the central hub for messages passed between each topic. Image Transport must be run here as it captures an image directly from a local camera. In the DIVER ROV system, the camera interfaces with the system via USB to the Raspberry Pi, so the Image Transport node must be run here and subsequently posts an image topic for use by the OpenTLD node on the surface station. The Autonomous Control node and Teleoperational Control nodes are run here to offload additional processing upon the Pi and free up the surface station to process the video feed. Figure 66 shows a brief overview of the control system.

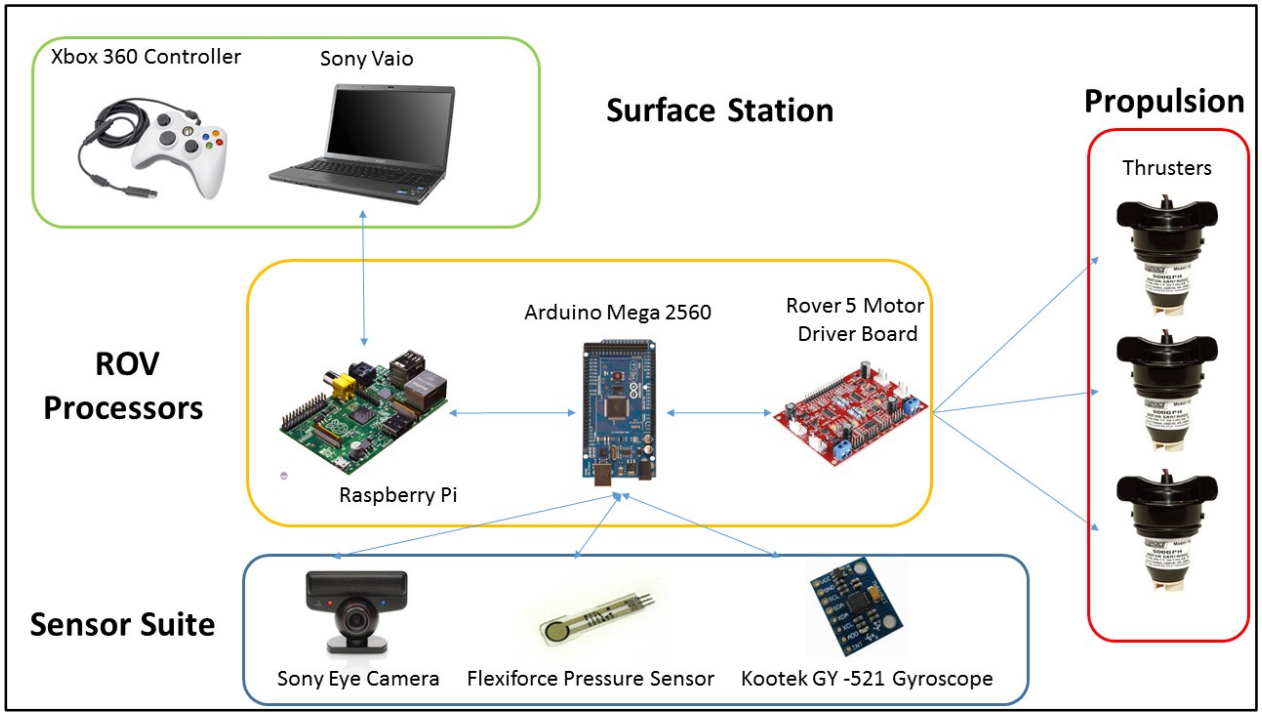


Figure 66: Control System Overview

The Arduino Mega is incapable of running nodes in the traditional sense of ROS. Instead, it uses a library called ROSSerial to publish and subscribe to topics. This emulates the functionality of a

traditional ROS node. The Arduino is responsible for translating the output signals of either the Autonomous Control node or the Teleoperational Control node running on the Pi and turning them into usable control signals for the motor control units. The Arduino is additionally responsible for reading data from the sensors, doing calculations on that data, and publishing the data for the rest of the control system. For a specific technical mapping of the control system, please refer to Appendix D: Electronic Schematics

CHAPTER 4: Conclusions

4. Conclusion

The DIVER ROV Team created a remotely operated vehicle capable of assisting, tracking and monitoring professional scuba diving teams. The DIVER ROV was designed to be easily deployable and lightweight, allowing diving teams to carry the system with other standard equipment. It is designed to facilitate tele-presence in underwater missions, where it is not possible or cost effective to deploy professional scuba divers. A portable surface station and integrated video game controller allows for ease of use and a short operator learning curve. By reducing the amount of personnel needed for an underwater mission, the DIVER ROV is able to decrease cost while maintaining awareness. Additionally, the DIVER ROV allows personnel, such as engineers, to participate in missions through the use of an onboard camera to provide real time video to the surface station. The DIVER ROV uses an open source, real time tracking algorithm known as OpenTLD. The algorithm allows the operator to select a target to track within a video frame rather than relying on previously developed tracking model. Additionally, the algorithm has the ability to understand the relative distance between the selected object using the variable area of the selected region in the video frame. Integrating the software with the DIVER ROV allows the surface operator to select and track a new model at the beginning of each mission. This function allows the surface operator to switch between tracked objects.

The DIVER ROV frame has six major components, each with a slotted design to allow ease of assembly and reduce manufacturing time. These components include the two side runners, four angled braces, two vertical braces, two front and two rear side thruster mounts and one center thruster mount. Stress on the DIVER ROV frame occurs as a result of the forward thrust and the opposing drag forces. These drag forces were calculated based on a cylindrical model of the DIVER ROV moving through water at a maximum speed of one meter per second. Based on this

estimation, the force and subsequent power required to overcome the drag of up to 31 Newtons when the DIVER ROV is traveling at maximum speed underwater was calculated. Using these results, the finite element method of stress analysis was used to determine stress concentrations on various components, namely the thruster mounts during normal operation. The results of the analysis showed that no structural failure of the frame's thruster mounts would occur at the maximum speed of one meter per second. Another objective of the DIVER ROV was to improve professional scuba diver safety and surface communication. The DIVER ROV facilitates visual communication between surface operators, such as engineers or boat pilots, and professional scuba divers underwater, thus creating a safer diving environment by incorporating a real time tracking and monitoring system. Communication between the surface and the underwater diver improved through the DIVER ROV's onboard LED communication system. The surface station operator can quickly send messages to the diver through a predetermined language such as Morse code.

The team faced project constraints throughout the design, fabrication and testing stages of the DIVER ROV project. Budget constraints limited the material and component selection as well as the ability to design custom electronics and investigate high performance components as viable options. The team was able to develop an attainable mission statement and create a design capable of meeting basic mission requirements. Complications in material tolerance and unfamiliarity with plastic machining techniques created complications in the fabrication stages of the project. External manufacturing tolerances for flat PMMA sheets presented inconsistences in component thickness. This affected the design of the slotted features and created frame instabilities where too high of a tolerance was present. Components were re-manufactured to account for tolerance differences and assembled.

Initial testing was conducted on the individual components to ensure manufacturer quality was met. This included testing motors for functionality and power draw, testing sensors for proper outputs and inspecting frame components for manufacturing quality. Once the team was confident that each component functioned properly, these components were integrated into the DIVER ROV. The next step was functional testing of the control system. The focus of this phase was to ensure that individual hardware components functioned together with the installed software. Specifically, the X-box controller was tested to ensure that the user inputs on the controller resulted in the desired outputs on the DIVER ROV, such as the thrusters rotating at the appropriate speeds. This testing phase allowed us to identify and troubleshoot software and hardware errors. Future recommendations for the DIVER ROV include a full qualification testing plan and the improvement of design of the DIVER ROV.

Proposed improvements to the DIVER ROV project include an upgrade of components such as the thrusters. The team used bilge pump motors due to the limited budget allowed for the project. Sponsorships could allow future ROV teams access to commercial grade thrusters which would provide the DIVER ROV with a more efficient propulsion system. With an increased budget more thrusters could be used. The current three-thruster configuration could be upgraded to a five or six thruster configuration, giving the DIVER ROV greater maneuverability. The team would also propose an expansion of the software and hardware hierarchy, allowing the integration of mission specific payloads. One example of this is a mechanical arm system to pick up and hold objects such as tools. Implementation of these payloads would increase the DIVER ROV's ability to assist, track and monitor professional scuba divers.

REFERENCES

[1] Andaloro, Franco, et al. "Assessing the suitability of a remotely operated vehicle (ROV) to study the fish community associated with offshore gas platforms in the Ionaian Sea: a comparative analysis with underwater visual censuses (UVCs)." Helgoland Marine Research (2013): 241+. <http://go.galegroup.com/ps/i.do?id=GALE%7CA331483427&v=2.1&u=mlin_c_worpoly&it=r&p=AONE&sw=w>.

[2] "Antenna." 2014. Bluefin Robotics. 2014. < http://www.bluefinrobotics.com/technology/antenna/>.

[3] Arehart-Treichel, Joan. "Deep-Sea Diving: Lessening the Health Hazards." Science News 109.18 (1976): 284-285. <http://www.jstor.org/stable/3961075>.

[4] Bay, Herbert, et al. "Speeded-Up Robust Features (SURF)." Computer Vision and Image Understanding, Vol 110, No. 3 (2008): 346-359.

[5] "Boat Design." n.d. boatdesign.net. 2014. <http://www.boatdesign.net/forums/attachments/materials/50870d1291586703-heat-treated-fiberglass-mat-p1010127.jpg>.

[6] "Boat Propeller Advice and Tips." n.d. Savvy Boater. 2014. <http://www.savvyboater.com/boat-propeller-advice-and-tips.aspx>.

[7] Boudergui, K, et al. "The use of alpha particle tagged neutrons for the inspection of objects on the sea floor for the presence of explosives." Nuclear Instruments and Methods of Physics Research (2013): 133+. Academic One File. <http://go.galegroup.com/ps/i.do?id=GALE%7CA317442521&v=2.1&u=mlin_c_worpoly&it=r&p=AONE&sw=w>.

[8] Bulten, NWH. "Numerical analysis of waterjet propulsion system." Thesis. 2006. <http://alexandria.tue.nl/extra2/200612081.pdf>.

[9] “Careers in Diving." *Professional Diving Academy*. Professional Diving Academy, 2012. Web. 28 Apr. 2014.

[10] CBS News. Facial recognition software tested countywide by law enforcement. 31 January 2014. 10 February 2014. <http://www.cbs8.com/story/24604243/facial-recognition-software-tested-countywide-by-law-enforcement>.

[11] Chen, F. L. and C. T. Su. "Vision-Based Automated Inspection System in Computer Integrated Manufacturing." The International Journal of Advanced Manufacturing Technology, Vol 11, Number 3 (1996): 206.

[12] “China Young Sun LED Technology Co., LTD.” <https://www.sparkfun.com/datasheets/Components/YSL-R547W2C-A13.pdf>

[13] Christ, Robert D and Robert L Wernli. The ROV Manual a User Guide to Observation-class Remotely Operated Vehicles. Amsterdam: Butterworth-Heinemann, 2007. Print.

[14] Common Interfaces of Deature Detectors. 31 December 2013. 15 February 2014. <http://docs.opencv.org/modules/features2d/doc/common_interfaces_of_feature_detectors.html>.

[15] "Computer Vision Research Group." 2010. COMVIS. 2014. <http://comvis.ciitlahore.edu.pk/images/anpr2.png>.

[16] "Control of Underwater Manipulators Mounted on an ROV Using Base Force." IEEE Xplore (2013). < http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=933117>.

[17] "Controlling the Manipulator of an Underwater ROV Using a Coarse Calibrated Pan/tilt Camera." IEEE Xplore (n.d.). <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=933042&tag=1>.

[18] Defense Advanced Projects Agency. IMAGE UNDERSTANDING FOR BATTLEFIELD AWARENESS. 1996. 16 February 2014. <http://www.fas.org/irp/program/process/iuba_pipfina.htm#AREA1>.

[19] "Detection and Tracking of Objects in Underwater Video." IEEE Xplore (n.d.). <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1315079>.

[20] Extreme Tech. Facebook's facial recognition software is now as accurate as the human brain, but what now? 9 March 2014. 11 March 2014. <http://www.extremetech.com/extreme/178777-facebooks-facial-recognition-software-is-now-as-accurate-as-the-human-brain-but-what-now>.

[21] "Federation of American Scientist. Fundamentals of Naval Weapons Systems." Chap 15. n.d. 11 February 2014. <http://www.fas.org/man/dod-101/navy/docs/fun/part15.htm>.

[22] "Flexiforce Pressure Sensor (25lbs) Quick Start." Sparkfun. (2014). <https://www.sparkfun.com/tutorials/389>

[23] "FlexiForce Sensor Demo Kit." n.d. Paralax Inc. 2014. <http://www.parallax.com/product/30056>.

[24] Forbes. Germany Is Freaking Out About Facebook's Facial Recognition Feature (Again). 8 August 2012. 11 February 2014. <http://www.forbes.com/sites/kashmirhill/2012/08/16/germany-is-freaking-out-about-facebooks-facial-recognition-feature-again/>.

[25] Gietler, Scott. "Backscatter Underwater." 2013. Underwater Photography Guide. 2014. <http://www.uwphotographyguide.com/backscatter-underwater>.

[26] Iteris. Detection Solutions. n.d. 11 February 2014. <http://www.iteris.com/solutions/detection>.

[27] Jaffe, J.S. "Computer modeling and the design of optimal underwater imaging systems." IEEE Journal of Oceanic Engineering (1990): 101, 111. <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=50695&isnumber=1853>.

[28] Jones, D.A. and D.B. Clarke. An Evaluation of the FIDAP Computational Fluid Dynamics Code for the Calculation of Hydrodynamic Forces on Underwater Platforms. Victoria, Australia: DTSI Okatfirns Sciences Laboratory, 2003. Document.

[29] “The Calculation of Hydrodynamic Coefficients for Underwater Vehicles.” Victoria, Australia: DTSO Platforms Sciences Laboratory, 2002. Document.

[30] Kalal, Dr. Zdenek. n.d. 2014.

[31] Kalal, Zdenek. "Tracking Learning Detection." Thesis. 2011.

[32] Kim, Jinhyun. Thruster Modeling and Controller Design for Unmanned Underwater Vehicles (UUVs). 2009. <http://www.intechopen.com/books/underwater_vehicles/thruster_modeling_and_controller_design_for_unmanned_underwater_vehicles__uuvs_>.

[33] Kohn, Natty and Shai Katz. "Signal and Image Processing Lab." April 2011. 2014. <figure 3: http://sipl.technion.ac.il/new/Teaching/Projects/Projects-Pages/20-2-08/html/Project%20web%20page_files/image008.png>.

[34] Kushner, David. "The Making of Arduino." IEEE Spectrum (2011). 04 March 2014.

[35] "Learning Control for Underwater Robotic Vehicles." IEEE Xplore (n.d.). <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=272779>.

[36] Long, John H, et al. "Four Flippers or Two? Tetrapodal Swimming with an Aquatic Robot." Bioinspiration & Biomimetics (2006): 20-29. Print.

[37] "LSM303DLMTR Breakout Board." n.d. Sparkfun. 2014. <https://www.sparkfun.com/products/10888>.

[38] Lynn, D.C. and G.S. Bohlander. "Performing ship hull inspections using a remotely operated vehicle." OCEANS '99 (1999): 555, 562. <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=804763>.

[39] MacIver, M.A., E Fontaine and J.W. Burdick. "Designing future underwater vehicles: principles and mechanisms of the weakly electric fish." IEEE Journal of Oceanic Engineering (2004): 651, 659. <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1353418&isnumber=29737>.

[40] Majumdar, Jyotsna D. "Underwater Welding - Present Status and Future Scope." Journal of Naval Architecture and Marine Engineering (2006): 39-47. Web. <http://www.freewebs.com/ejname/j31/p38_47.pdf>.

[41] Massey. Mechanics of Fluids. 4th. 1979. Textbook.

[42] Meinecke, G, V Ratmeyer and J Renken. "HYBRID-ROV - Development of a new underwater vehicle for high-risk areas." OCEANS 2011 (2011): 1, 6.

[43] "MiniRover." 2014. Teledyne Benthos. 2014. <http://www.benthos.com/index.php/product/remotely_operated_vehicles/minirover>.

[44] Molchan, Marianne. "The Role of Micro-ROVs in Maritime Safety and Security." Molchan Marine Sciences, USA (2005). <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.119.3003&rep=rep1&type=pdf>.

[45] "Motor Driver 15A IRF7862PBF." n.d. Sparkfun. 2014. <https://www.sparkfun.com/products/9107>.

[46] "Multivariable Self-tuning Autopilots for Autonomous and Remotely Operated Underwater Vehicles." IEEE Explore (2013). <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=107142>.

[47] Nave, R. "Buoyancy." *Pressure*. Hyperphysics, n.d. Web. 25 Apr. 2014.

[48] Nave, R. "Pascal's Principle." *Pressure*. Hyperphysics, n.d. Web. 26 Apr. 2014.

[49] Oloffson, Niclas and A. B. Rolls-Royce. "The Specialist Committee on Validation of Waterjet Test Procedures." ITTC (n.d.): 387-414. <http://ittc.sname.org/proc23/Waterjet.pdf>.

[50] OpenCV. Cascade Classification. 21 December 2013. 11 February 2014. <http://docs..org/modules/objdetect/doc/cascade_classification.html>.

[51] "OtterBox 3000 Cases." n.d. Otter Box. 2014. <http://www.otterbox.com/otterbox-3000-cases/otterbox-3000-cases,default,sc.html>.

[52] Papageorgiou, Oren and Poggio. "A general framework for object detection." International Conference on Computer Vision. 1998.

[53] "Photobucket." n.d. < http://s104.photobucket.com/user/GOD>

[54] “PlayStation Eye”. Sony. Amazon. (2014). <http://ecx.images-amazon.com/images/I/51ALDcK--SL._SY300_.jpg>

[55] Prayogo, N, et al. "Design and testing of underwater thruster for shrimp ROV -itb." Indian Journal of Marine Sciences (2009): 339-345.

[56] “Pressure Increase with Ocean Depth.” National Ocean Service. National Oceanic and Atmospheric Administration. NOAA. (2014). <http://oceanservice.noaa.gov/facts/pressure.html>

[57] "Programming - Computer Vision Tutorial." n.d. Society of Robots. 2014.

[58] Rambhia, Jay. "Tracking in OpenTLD aka Predator." 05 June 2012. Jay Rambhia. 2014. <http://jayrambhia.com/assets/images/predator-1.jpg>.

[59] "Raspberry Pi – Model B." n.d. Sparkfun. 2014. <https://www.sparkfun.com/products/11546 >.

[60] Read, Dave. "Aiming Strobes." n.d. daveread.com. 2014. <http://www.daveread.com/uw-photo/drawings/backscatter1.gif>.

[61] "Robot for Research and Innovation." n.d. Willow Garage. 2014. <http://www.willowgarage.com/pages/pr2/overview>.

[62] Simpson, J. A., B L Hughes and J F Muth. "Smart Transmitters and Receivers for Underwater Free-Space Optical Communication." Selected Areas in Communications (2012). <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6214706&queryText%3Dunderwater+communication>.

[63] “Sony VAIO F Series VPC-F115FM/B Review.” I-techtalk. (2014). <http://blog.itechtalk.com/wp-content/2010/04/125.png>

[64] Soylu, S, B J Buckham and R P Podhorodeski. "Dynamics and Control of Tethered Underwater-Manipulator Systems." OCEANS 2010 (2010): 1-8. <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=5664366&ranges%3D2010_2013_p_Publication_Year%26queryText%3Dtethered+vehicles>.

[65] Stanford University. Army Robots. n.d. 11 February 2014. <http://cs.stanford.edu/people/eroberts/cs181/projects/2010-11/ComputersMakingDecisions/army-robots/index.html>.

[66] "Swimming Socket Wrenches." n.d. American Oil & Gas. 2014. <http://aoghs.org/offshore-exploration/rovs-swimming-wrenches/>.

[67] Taylor, Mikell. "Why Autonomous Gliders Are the Hot New Ocean Technology."*IEEE Spectrum*. IEEE, 5 Sept. 2008. Web. 26 Apr. 2014.

[68] Thone, Stephen. "Mayfair 750 GPH Bilge Pump Thruster Testing." 2009. Homebuild ROV's. 2014. <http://www.homebuiltrovs.com/mayfair750test.html>.

[69] Todnem, K., et al. "Neurological Long Term Consequences of Deep Diving." British Journal of Industrial Medicine (1991): 258-266. <http://www.jstor.org/stable/27727232>.

[70] Traxxas Propeller. A Main Hobbies. (2014). <http://www.amain.com/images/large/tra/tra1583.jpg>

[71] "Underwater ROV." 2014. Nautic Expo. 2014. <http://www.nauticexpo.com/prod/videoray/underwater-rovs-maximum-intervention-depth-300-m-23504-269637.html>.

[72] Valdivia, A, et al. "A Soft Body Under-actuated Approach to Multi Degree of Freedom Biomimetic Robots: A Stingray Example." Biomedical Robotics and Biomechatronics (BioRob) (2010): 473-478.

[73] "Versatrax 300." n.d. Inuktun. 2014. <http://www.inuktunusa.com/crawler-vehicles/versatrax-300.html >.

[74] Viola and Jones. "Rapid object detection using a boosted cascade of simple features." Computer Vision and Pattern Recognition (2001).

[75] Walker, Daniel G. "Design and Control of a High Maneuverability Remotely Operated Vehicle with Multi-Degree of Freedom Thrusters." Research. 2005. Print.

[76] Walther, D, D. R Edgington and C Koch. "Detection and tracking of objects in underwater video." Computer Vision and Pattern Recognition. IEEE Computer Society Conference, 2004.

[77] Wasserman, K S, et al. "Dynamic buoyancy control of an ROV using a variable ballast tank." OCEANS 2003. 2003.

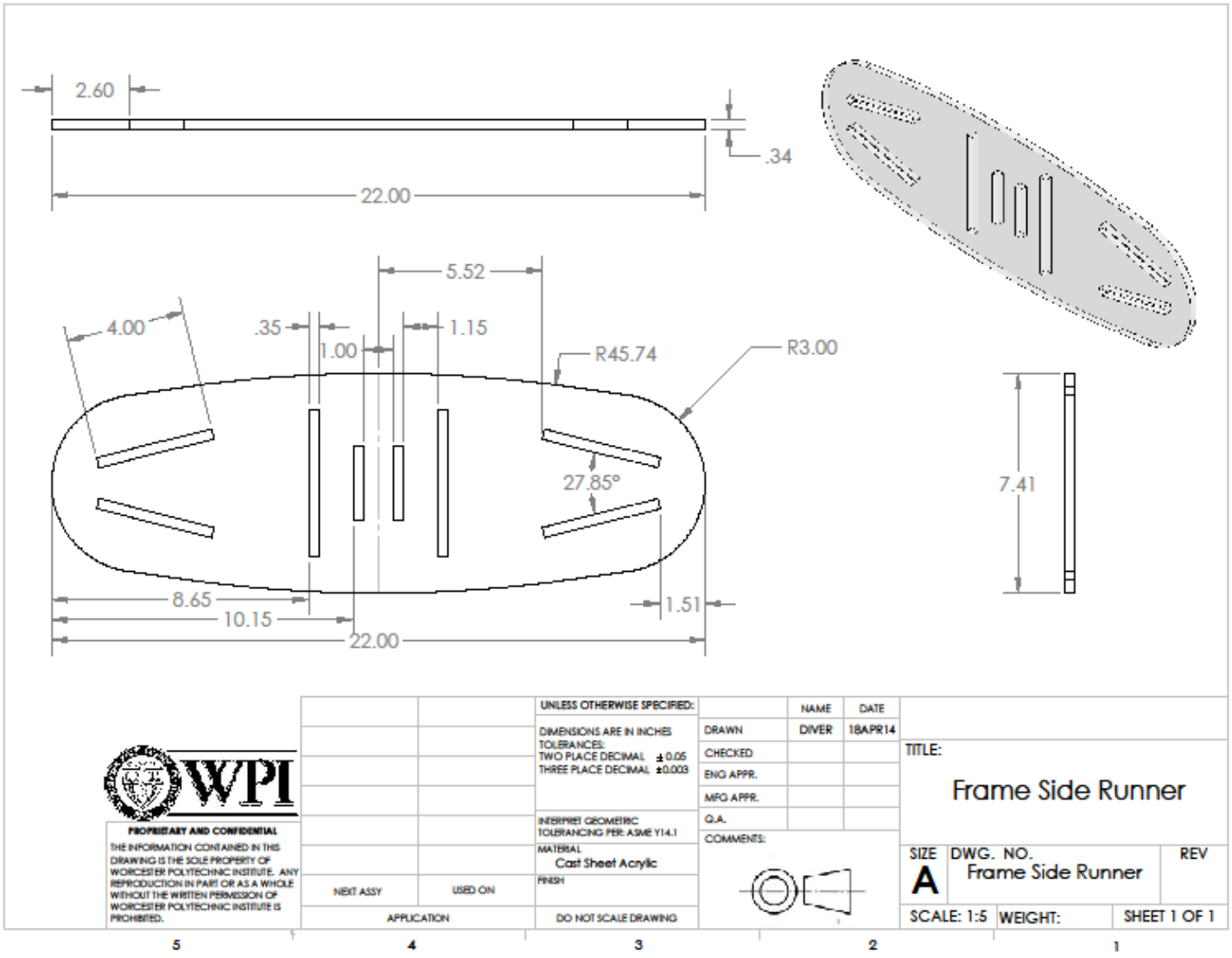
[78] Woods, John W. Multidimensional Signal, Image, and Video Processing and Coding. Amsterdam, 2011. Safari Books Online.

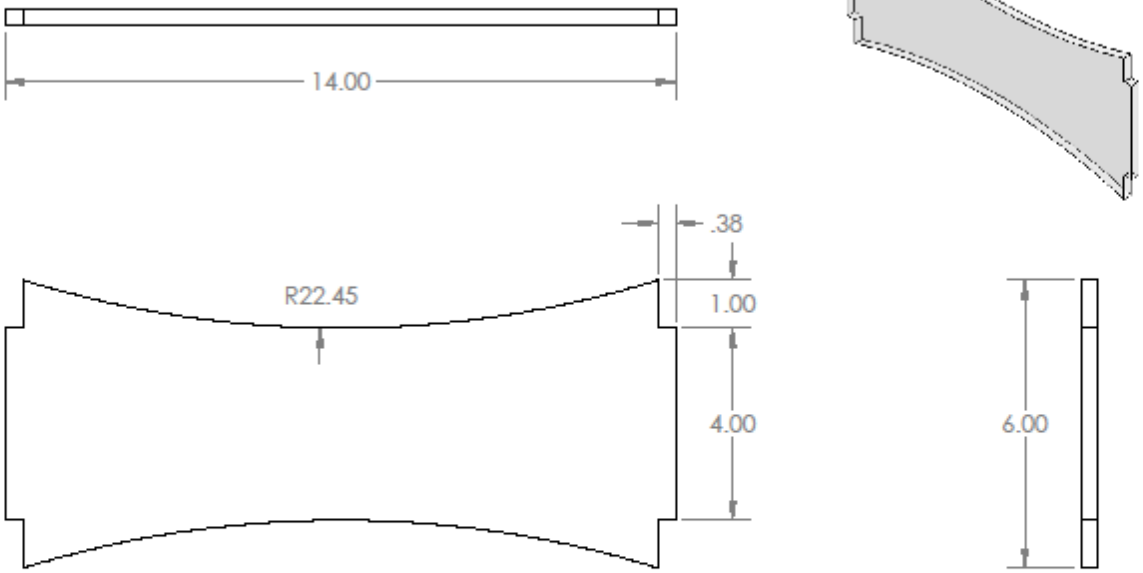
[79] Wu, Jian, et al. "A Comparative Study of SIFT and its Variants." Measurement Science Review, Volume 13, No. 3 (2013): 122-131.

[80] Yangwei, Wang, Wang Zhenlong and Li Jian. "Initial design of a Biomimetic Cuttlefish Robot Actuated by SMA Wires." Measuring Technology and Mechatronics Automation (ICMTMA), 2011 Third International Conference. 2011.

APPENDICES

Appendix A: DIVER Part Drawings





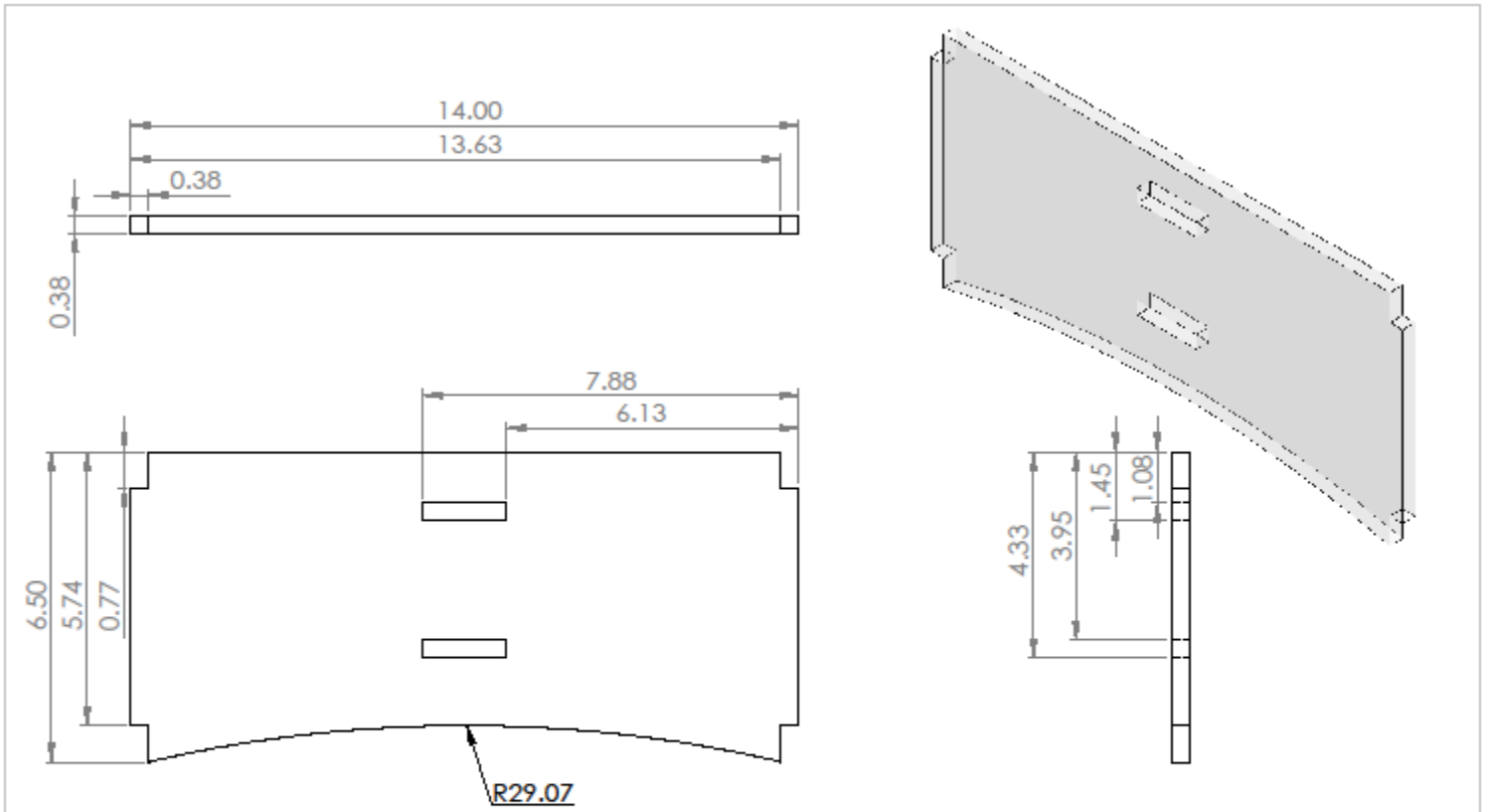
		UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES TOLERANCES: TWO PLACE DECIMAL ±0.05 THREE PLACE DECIMAL ±0.003	DRAWN	DIVER	18APR14
			CHECKED		
			ENG APPR.		
			MFG APPR.		
		INTERPRET GEOMETRIC TOLERANCING PER: ASME Y14.1	Q.A.		
		MATERIAL Cast Sheet Acrylic	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

WPI

TITLE: Frame Support Angled

SIZE	DWG. NO.	REV
A	Frame Support Angled	
SCALE: 1:3	WEIGHT:	SHEET 1 OF 1

5 4 3 2 1



WPI

PROPRIETARY AND CONFIDENTIAL

THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF WORCESTER POLYTECHNIC INSTITUTE. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF WORCESTER POLYTECHNIC INSTITUTE IS PROHIBITED.

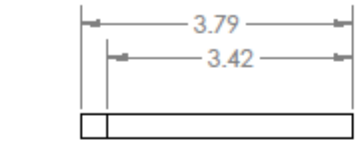
UNLESS OTHERWISE SPECIFIED:		NAME	DATE
DIMENSIONS ARE IN INCHES	DRAWN	DIVER	18APR14
TOLERANCES:	CHECKED		
TWO PLACE DECIMAL ±0.05	ENG APPR.		
THREE PLACE DECIMAL ±0.003	MFG APPR.		
INTERPRET GEOMETRIC TOLERANCING PER: ASME Y14.1	Q.A.		
MATERIAL: Cast Sheet Acrylic	COMMENTS:		
FINISH			
NEXT ASSY / USED ON			
APPLICATION	DO NOT SCALE DRAWING		

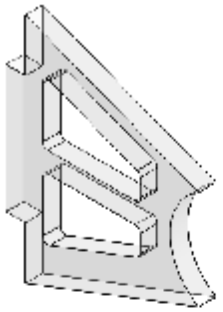
TITLE: Vertical Center Frame Support

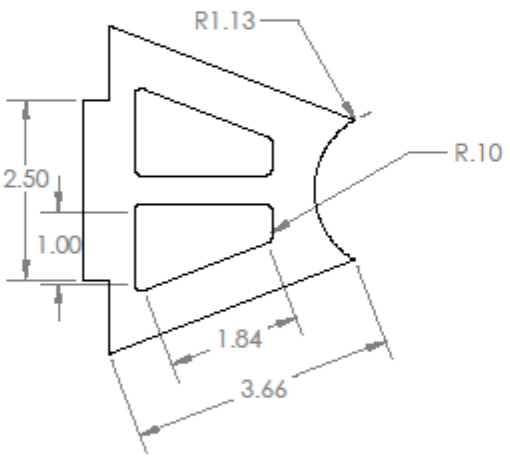
SIZE	DWG. NO.	REV
A	Frame Support Veritcal Center	

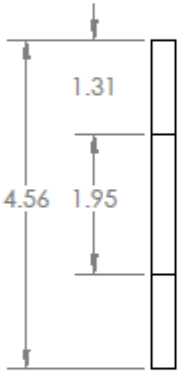
SCALE: 1:3 | WEIGHT: | SHEET 1 OF 1

5 4 3 2 1









WPI

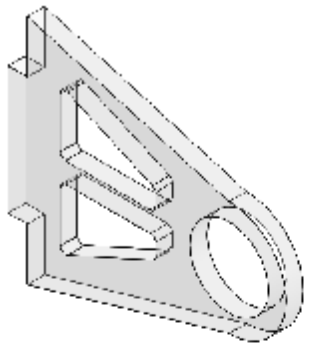
NEXT ASSY	USED ON	UNLESS OTHERWISE SPECIFIED:		NAME	DATE
		DIMENSIONS ARE IN INCHES TOLERANCES: TWO PLACE DECIMAL ±0.05 THREE PLACE DECIMAL ±0.003	DRAWN	DIVER	18APR14
			CHECKED		
			ENG APPR.		
			MFG APPR.		
			Q.A.		
		INTERPRET GEOMETRIC TOLERANCING PER: ASME Y14.1	COMMENTS:		
		MATERIAL Cast Sheet Acrylic			
		FINISH			
APPLICATION		DO NOT SCALE DRAWING			

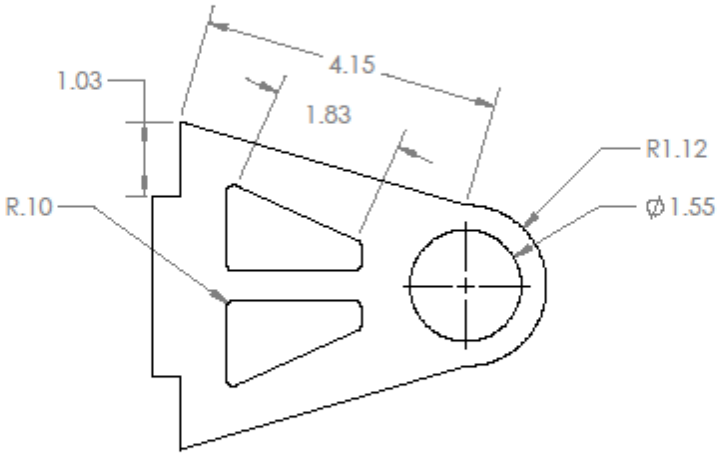
TITLE:

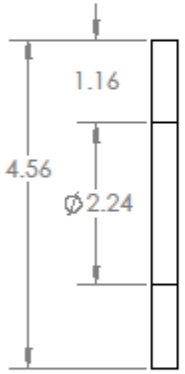
Thruster Mounting Frame Front

SIZE	DWG. NO.	REV
A	Front Thruster Mounting Frame	
SCALE: 1:2	WEIGHT:	SHEET 1 OF 1

5 4 3 2 1







WPI

PROPRIETARY AND CONFIDENTIAL

THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF WORCESTER POLYTECHNIC INSTITUTE. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF WORCESTER POLYTECHNIC INSTITUTE IS PROHIBITED.

NEXT ASSY	USED ON	
APPLICATION		

UNLESS OTHERWISE SPECIFIED:

DIMENSIONS ARE IN INCHES
TOLERANCES:
TWO PLACE DECIMAL ±0.05
THREE PLACE DECIMAL ±0.003

INTERPRET GEOMETRIC TOLERANCING PER: ASME Y14.1

MATERIAL
Cast Sheet Acrylic

FINISH

DO NOT SCALE DRAWING

	NAME	DATE
DRAWN	DIVER	18APR14
CHECKED		
ENG APPR.		
MFG APPR.		
Q.A.		

COMMENTS:

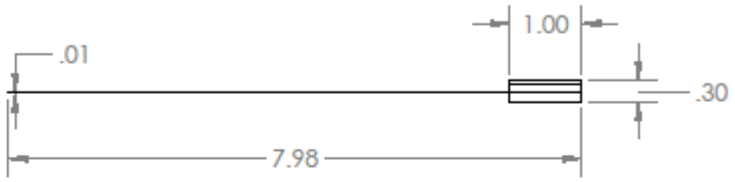
TITLE:

Rear Thruster Mounting Frame

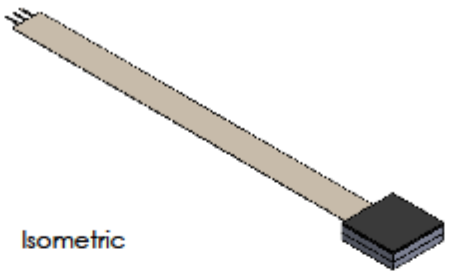
SIZE	DWG. NO.	REV
A	Thruster Mounting Frame Rear	

SCALE: 1:2 | WEIGHT: | SHEET 1 OF 1

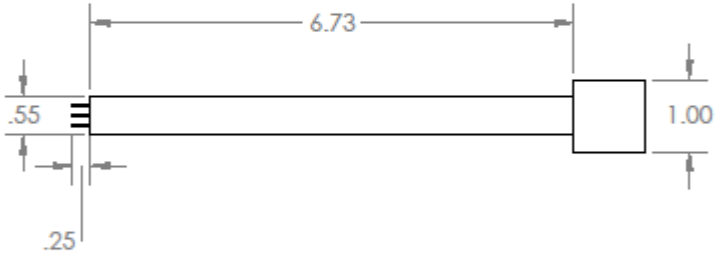
5 4 3 2 1



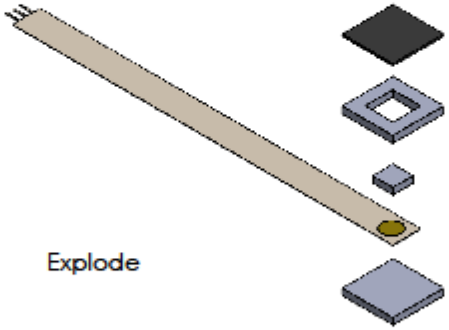
Right



Isometric



Top



Explode

WPI

NEXT ASSY	USED ON	
APPLICATION		

UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES
TOLERANCES:
TWO PLACE DECIMAL ±0.05
THREE PLACE DECIMAL ±0.003
INTERPRET GEOMETRIC TOLERANCING PER: ASME Y14.1
MATERIAL
FINISH
DO NOT SCALE DRAWING

	NAME	DATE
DRAWN	DIVER	18APR14
CHECKED		
ENG APPR.		
MFG APPR.		
Q.A.		

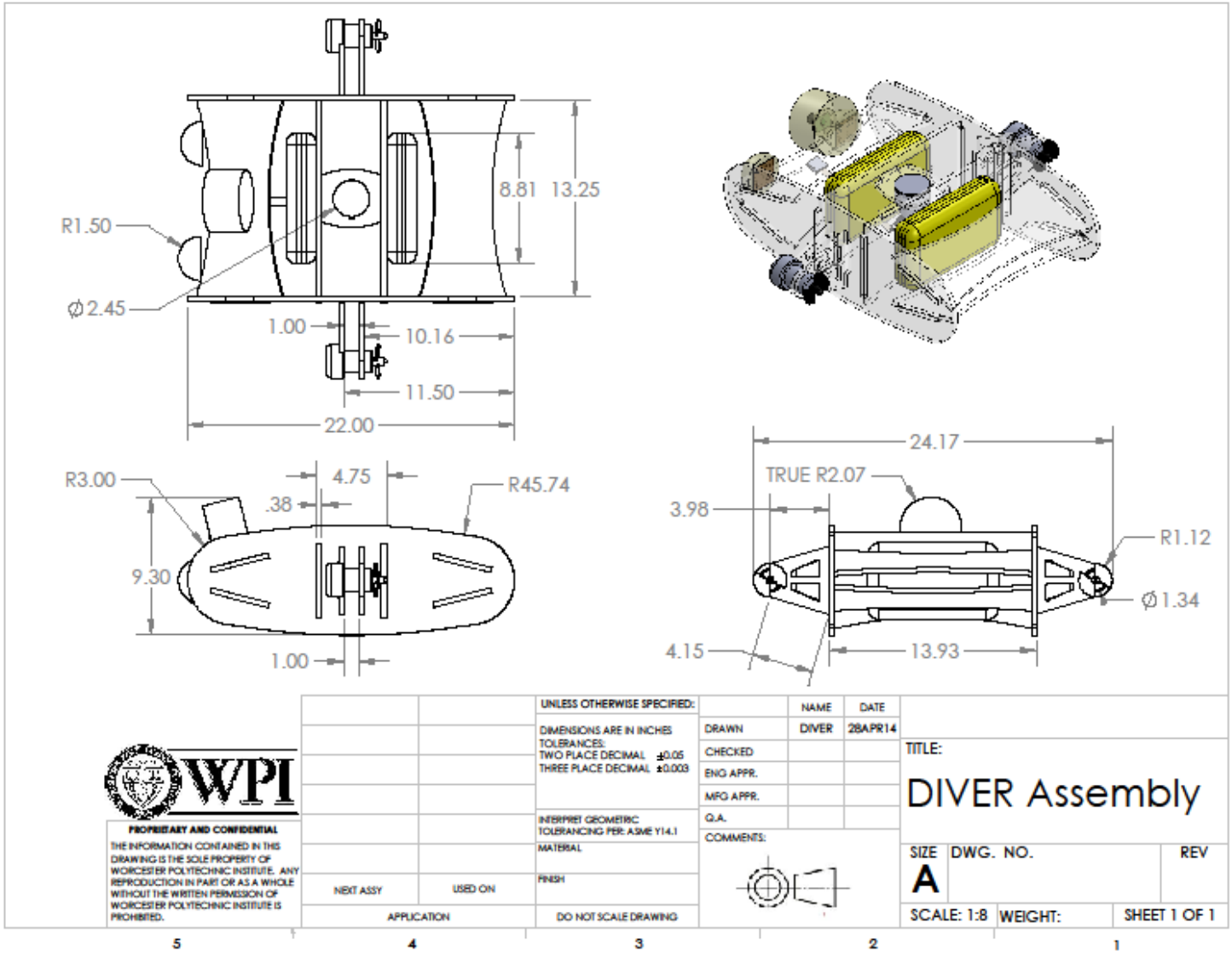
COMMENTS:

TITLE:
Pressure Sensor Housing Assembly

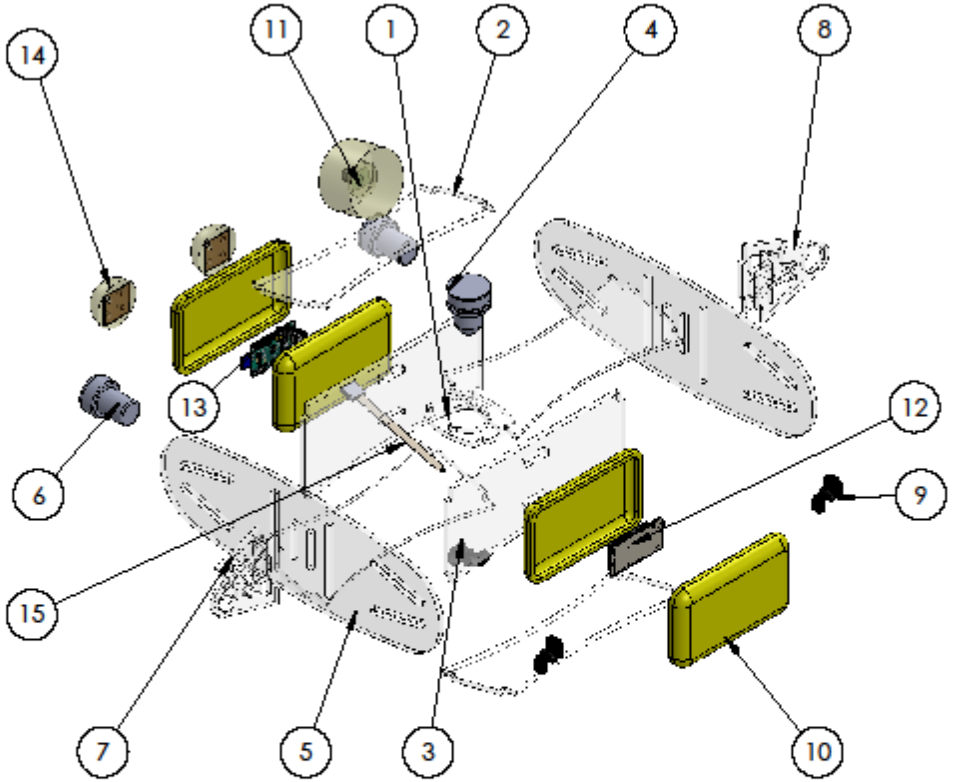
SIZE	DWG. NO.	REV
A	Pressure Sensor Housing Assembly	
SCALE: 1:2	WEIGHT:	SHEET 1 OF 1

5 4 3 2 1

Appendix B: DIVER Assembly



ITEM NO.	PART NUMBER	QTY.
1	Center Motor Brace	1
2	Frame Support Angled	4
3	Frame Support Veritcal Center	2
4	Vertical Thruster Motor	1
5	Frame Side Runner	2
6	Side Thruster Motor	2
7	Front Thruster Mount	2
8	Rear Thruster Mount	2
9	Traxis Propeller	3
10	Otterbox Drybox 3000	2
11	Camera Assembly	1
12	Arduino Mega	1
13	Raspberry Pi Assembly	1
14	LED Array Assembly	2
15	Pressure Sensor Housing	1



WPI

NEXT ASSY | USED ON | APPLICATION

UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES
TOLERANCES:
TWO PLACE DECIMAL ±0.05
THREE PLACE DECIMAL ±0.003
INTERPRET GEOMETRIC TOLERANCING PER: ASME Y14.1
MATERIAL
FINISH
DO NOT SCALE DRAWING

	NAME	DATE
DRAWN	DIVER	4/28/14
CHECKED		
ENG APPR.		
MFG APPR.		
Q.A.		

COMMENTS:

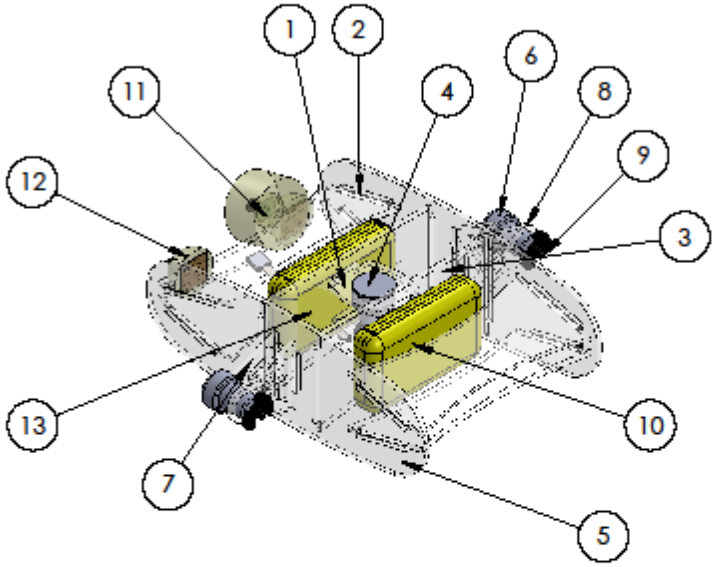
TITLE:
DIVER Assembly

SIZE A | DWG. NO. | REV

SCALE: 1:8 | WEIGHT: | SHEET 1 OF 1

5 4 3 2 1

ITEM NO.	PART NUMBER	QTY.
1	Center Motor Brace	1
2	Frame Support Angled	4
3	Frame Support Veritcal Center	2
4	Vertical Thruster Motor	1
5	Frame Side Runner	2
6	Side Thruster Motor	2
7	Front Thruster Mount	2
8	Rear Thruster Mount	2
9	Traxis Propeller	3
10	Otterbox Drybox 3000	2
11	Camera Assembly	1
12	LED Array Assembly	2
13	Pressure Sensor Housing	1



WPI

NEXT ASSY | USED ON

APPLICATION

UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES
TOLERANCES:
TWO PLACE DECIMAL ±0.05
THREE PLACE DECIMAL ±0.003
INTERPRET GEOMETRIC TOLERANCING PER: ASME Y14.1
MATERIAL
FINISH
DO NOT SCALE DRAWING

	NAME	DATE
DRAWN	DIVER	4/28/14
CHECKED		
ENG APPR.		
MFG APPR.		
Q.A.		

COMMENTS:

TITLE:

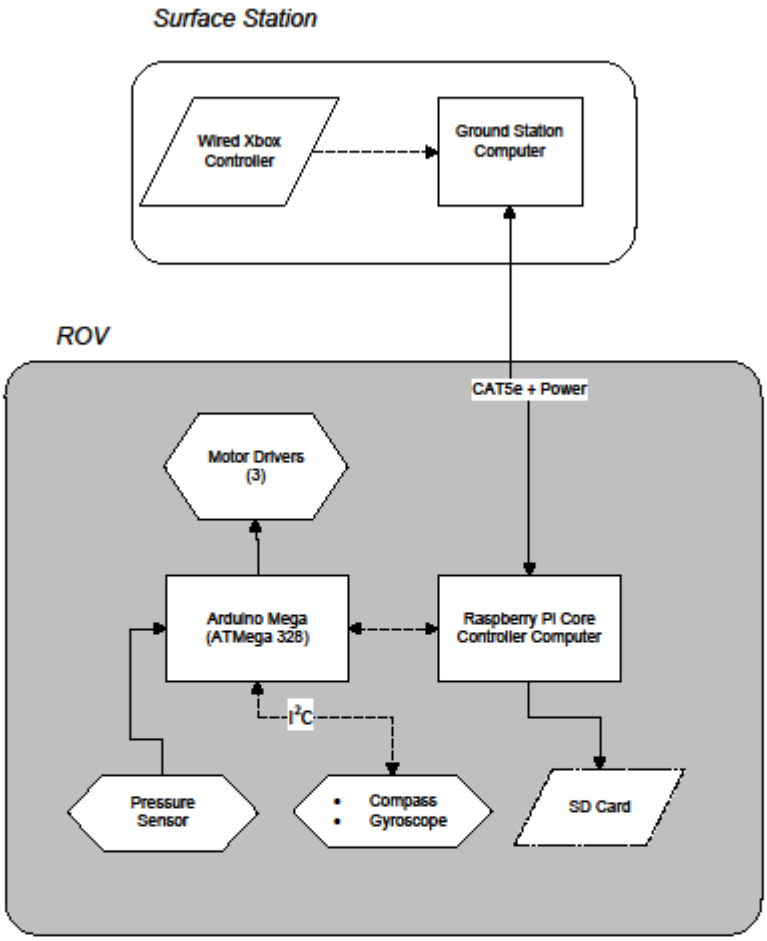
DIVER Assembly

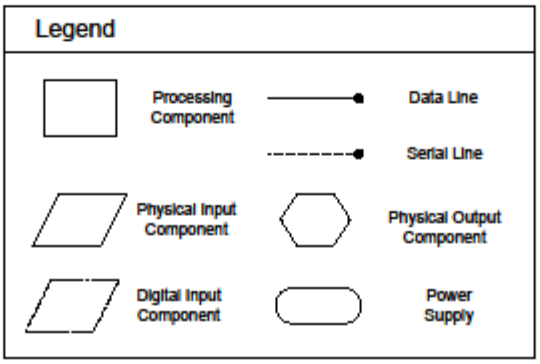
SIZE	DWG. NO.	REV
A		

SCALE: 1:8 | WEIGHT: | SHEET 1 OF 1

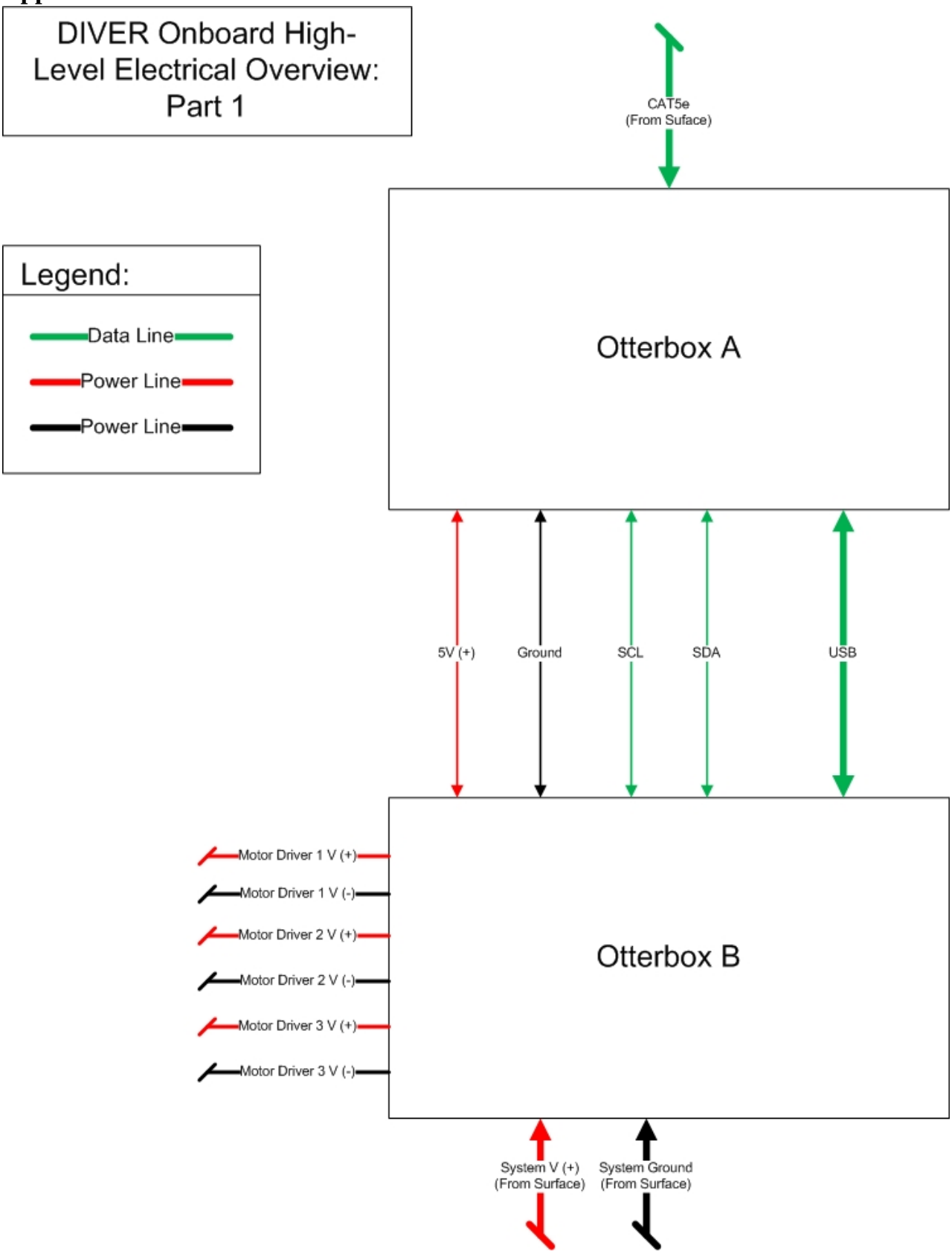
5 4 3 2 1

Appendix C: Control Overviews

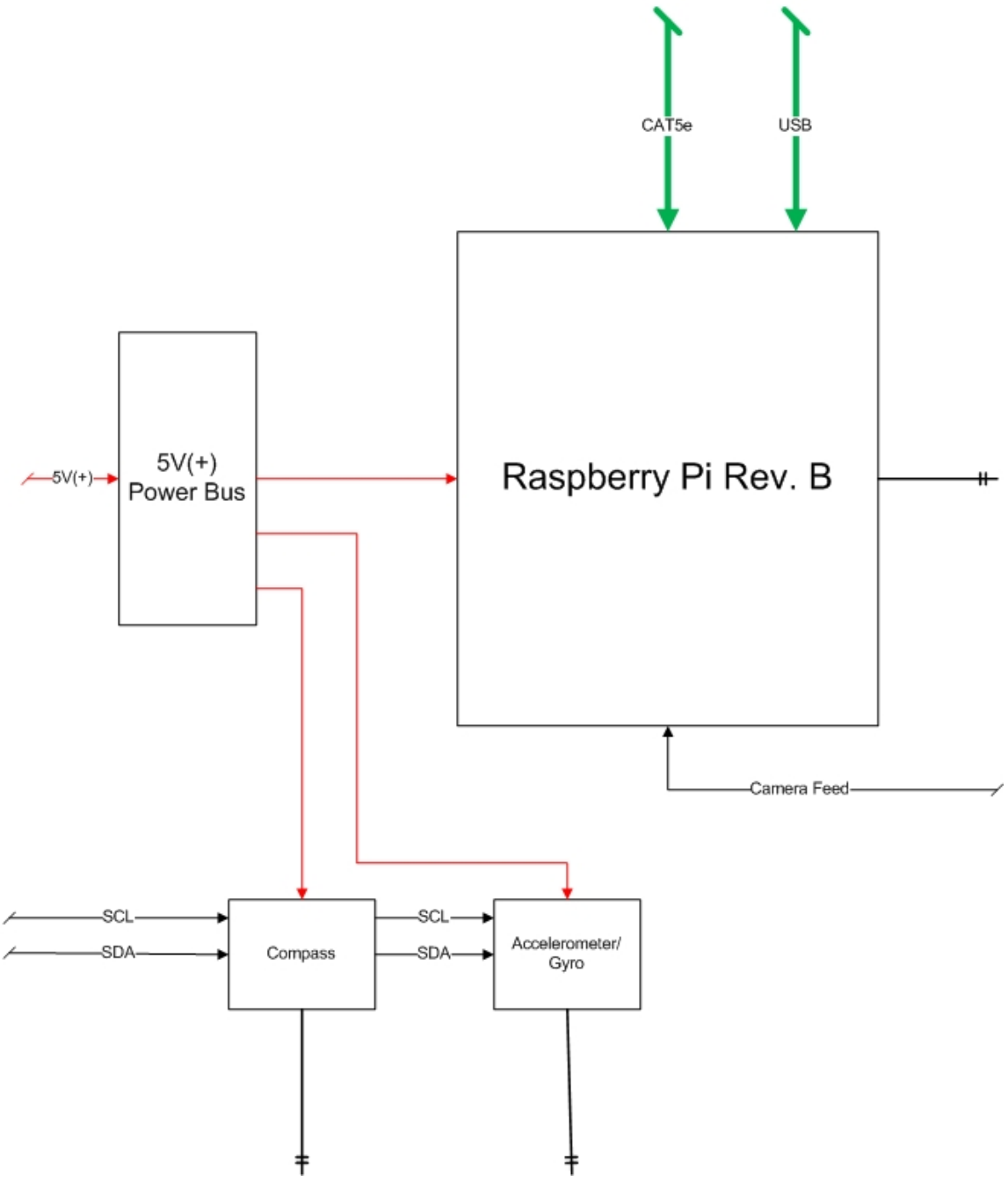


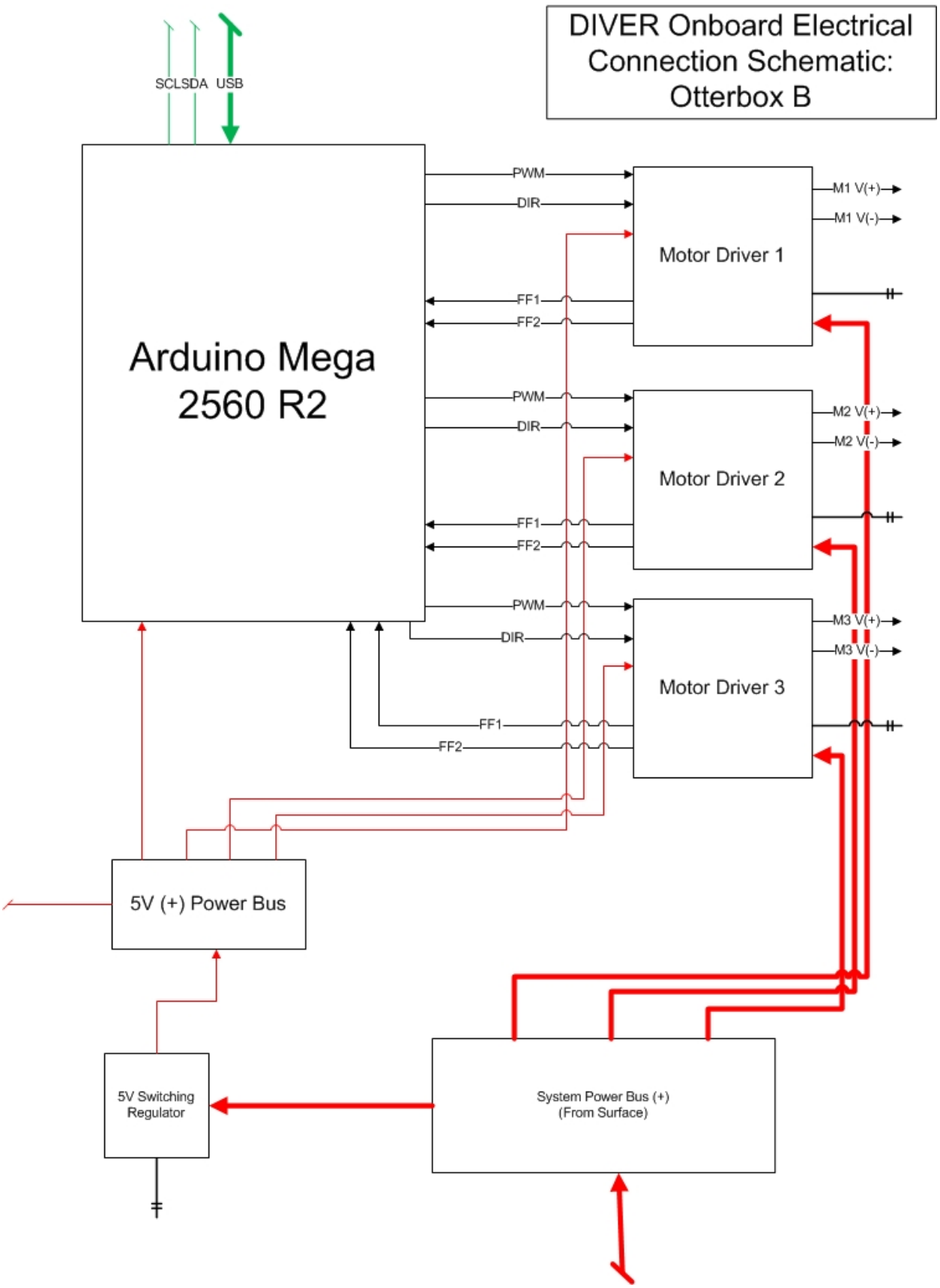


Appendix D: Electronic Schematics



DIVER Onboard Electrical Connection Schematic: Otterbox A




DIVER Onboard Electrical Connection Schematic: Otterbox B
SCL
SDA
USB
Arduino Mega 2560 R2
PWM
DIR
FF1
FF2
Motor Driver 1
M1 V(+)
M1 V(-)
Motor Driver 2
M2 V(+)
M2 V(-)
Motor Driver 3
M3 V(+)
M3 V(-)
5V (+) Power Bus
5V Switching Regulator
System Power Bus (+) (From Surface)

Appendix E: Code

Overview

In the interest of furthering academic growth, the DIVER team has chosen to designate the code of the DIVER robot as open source material. As such, the code is currently hosted in a GitHUB repository so that it may be accessed by any who wish to view or expand upon our work. Please refer to the following link for the code of the DIVER system.

https://github.com/chconley/DIVER

Although a complete listing of the source code is impractical, several code samples have been selected for reference. These are the absolute minimum required to operate the DIVER ROV and are listed below:

- arduinoFinal.ino
- teleMotor.py
- autoMotor.py
- base_frame.hpp
- base_frame.cpp
- main.hpp
- main.cpp

arduinoFinal.ino

```
#include <ros.h>
#include <Wire.h>
#include <SPI.h>
#include <std_msgs/Int16.h>
#include <std_msgs/String.h>
#include <I2Cdev.h>
#include <MPU60X0.h>
#include <LSM303.h>


#define MPU6050_I2C_ADDRESS 0x68

#define MPU6050_AUX_VDDIO          0x01   // R/W
#define MPU6050_SMPLRT_DIV         0x19   // R/W
#define MPU6050_CONFIG             0x1A   // R/W
#define MPU6050_GYRO_CONFIG        0x1B   // R/W
#define MPU6050_ACCEL_CONFIG       0x1C   // R/W
#define MPU6050_FF_THR             0x1D   // R/W
#define MPU6050_FF_DUR             0x1E   // R/W
#define MPU6050_MOT_THR            0x1F   // R/W
#define MPU6050_MOT_DUR            0x20   // R/W
#define MPU6050_ZRMOT_THR          0x21   // R/W
#define MPU6050_ZRMOT_DUR          0x22   // R/W
#define MPU6050_FIFO_EN            0x23   // R/W
#define MPU6050_I2C_MST_CTRL       0x24   // R/W
#define MPU6050_I2C_SLV0_ADDR      0x25   // R/W
#define MPU6050_I2C_SLV0_REG       0x26   // R/W
#define MPU6050_I2C_SLV0_CTRL      0x27   // R/W
#define MPU6050_I2C_SLV1_ADDR      0x28   // R/W
#define MPU6050_I2C_SLV1_REG       0x29   // R/W
#define MPU6050_I2C_SLV1_CTRL      0x2A   // R/W
```

```
#define MPU6050_I2C_SLV2_ADDR      0x2B   // R/W
#define MPU6050_I2C_SLV2_REG       0x2C   // R/W
#define MPU6050_I2C_SLV2_CTRL      0x2D   // R/W
#define MPU6050_I2C_SLV3_ADDR      0x2E   // R/W
#define MPU6050_I2C_SLV3_REG       0x2F   // R/W
#define MPU6050_I2C_SLV3_CTRL      0x30   // R/W
#define MPU6050_I2C_SLV4_ADDR      0x31   // R/W
#define MPU6050_I2C_SLV4_REG       0x32   // R/W
#define MPU6050_I2C_SLV4_DO        0x33   // R/W
#define MPU6050_I2C_SLV4_CTRL      0x34   // R/W
#define MPU6050_I2C_SLV4_DI        0x35   // R
#define MPU6050_I2C_MST_STATUS     0x36   // R
#define MPU6050_INT_PIN_CFG        0x37   // R/W
#define MPU6050_INT_ENABLE         0x38   // R/W
#define MPU6050_INT_STATUS         0x3A   // R
#define MPU6050_ACCEL_XOUT_H       0x3B   // R
#define MPU6050_ACCEL_XOUT_L       0x3C   // R
#define MPU6050_ACCEL_YOUT_H       0x3D   // R
#define MPU6050_ACCEL_YOUT_L       0x3E   // R
#define MPU6050_ACCEL_ZOUT_H       0x3F   // R
#define MPU6050_ACCEL_ZOUT_L       0x40   // R
#define MPU6050_TEMP_OUT_H         0x41   // R
#define MPU6050_TEMP_OUT_L         0x42   // R
#define MPU6050_GYRO_XOUT_H        0x43   // R
#define MPU6050_GYRO_XOUT_L        0x44   // R
#define MPU6050_GYRO_YOUT_H        0x45   // R
#define MPU6050_GYRO_YOUT_L        0x46   // R
#define MPU6050_GYRO_ZOUT_H        0x47   // R
#define MPU6050_GYRO_ZOUT_L        0x48   // R
#define MPU6050_EXT_SENS_DATA_00   0x49   // R
#define MPU6050_EXT_SENS_DATA_01   0x4A   // R
#define MPU6050_EXT_SENS_DATA_02   0x4B   // R
#define MPU6050_EXT_SENS_DATA_03   0x4C   // R
```

```
#define MPU6050_EXT_SENS_DATA_04   0x4D   // R
#define MPU6050_EXT_SENS_DATA_05   0x4E   // R
#define MPU6050_EXT_SENS_DATA_06   0x4F   // R
#define MPU6050_EXT_SENS_DATA_07   0x50   // R
#define MPU6050_EXT_SENS_DATA_08   0x51   // R
#define MPU6050_EXT_SENS_DATA_09   0x52   // R
#define MPU6050_EXT_SENS_DATA_10   0x53   // R
#define MPU6050_EXT_SENS_DATA_11   0x54   // R
#define MPU6050_EXT_SENS_DATA_12   0x55   // R
#define MPU6050_EXT_SENS_DATA_13   0x56   // R
#define MPU6050_EXT_SENS_DATA_14   0x57   // R
#define MPU6050_EXT_SENS_DATA_15   0x58   // R
#define MPU6050_EXT_SENS_DATA_16   0x59   // R
#define MPU6050_EXT_SENS_DATA_17   0x5A   // R
#define MPU6050_EXT_SENS_DATA_18   0x5B   // R
#define MPU6050_EXT_SENS_DATA_19   0x5C   // R
#define MPU6050_EXT_SENS_DATA_20   0x5D   // R
#define MPU6050_EXT_SENS_DATA_21   0x5E   // R
#define MPU6050_EXT_SENS_DATA_22   0x5F   // R
#define MPU6050_EXT_SENS_DATA_23   0x60   // R
#define MPU6050_MOT_DETECT_STATUS  0x61   // R
#define MPU6050_I2C_SLV0_DO        0x63   // R/W
#define MPU6050_I2C_SLV1_DO        0x64   // R/W
#define MPU6050_I2C_SLV2_DO        0x65   // R/W
#define MPU6050_I2C_SLV3_DO        0x66   // R/W
#define MPU6050_I2C_MST_DELAY_CTRL 0x67   // R/W
#define MPU6050_SIGNAL_PATH_RESET  0x68   // R/W
#define MPU6050_MOT_DETECT_CTRL    0x69   // R/W
#define MPU6050_USER_CTRL          0x6A   // R/W
#define MPU6050_PWR_MGMT_1         0x6B   // R/W
#define MPU6050_PWR_MGMT_2         0x6C   // R/W
#define MPU6050_FIFO_COUNTH        0x72   // R/W
#define MPU6050_FIFO_COUNTL        0x73   // R/W
```

```
#define MPU6050_FIFO_R_W           0x74   // R/W
#define MPU6050_WHO_AM_I           0x75   // R

struct t_mtab { char c, pat; } ;

struct t_mtab morsetab[] = {
      {'.', 106},
      {',', 115},
      {'?', 76},
      {'/', 41},
      {'A', 6},
      {'B', 17},
      {'C', 21},
      {'D', 9},
      {'E', 2},
      {'F', 20},
      {'G', 11},
      {'H', 16},
      {'I', 4},
      {'J', 30},
      {'K', 13},
      {'L', 18},
      {'M', 7},
      {'N', 5},
      {'O', 15},
      {'P', 22},
      {'Q', 27},
      {'R', 10},
      {'S', 8},
      {'T', 3},
      {'U', 12},
      {'V', 24},
      {'W', 14},
```

```
      {'X', 25},
      {'Y', 29},
      {'Z', 19},
      {'1', 62},
      {'2', 60},
      {'3', 56},
      {'4', 48},
      {'5', 32},
      {'6', 33},
      {'7', 35},
      {'8', 39},
      {'9', 47},
      {'0', 63}
} ;

#define N_MORSE  (sizeof(morsetab)/sizeof(morsetab[0]))

#define SPEED  (12)
#define DOTLEN  (1200/SPEED)
#define DASHLEN  (3*(1200/SPEED))

LSM303 compass;
MPU60X0 accelgyro;

int LEDpin = 13 ;

String morseMsg;

int16_t ax, ay, az;
int16_t gx, gy, gz;

int sensorReading = 0;
```

```
ros::NodeHandle  nh;

std_msgs::Int16 x_tilt_msg;
ros::Publisher pub_tilt_x("attitude_x", &x_tilt_msg);
std_msgs::Int16 y_tilt_msg;
ros::Publisher pub_tilt_y("attitude_y", &y_tilt_msg);
std_msgs::Int16 z_tilt_msg;
ros::Publisher pub_tilt_z("attitude_z", &z_tilt_msg);

std_msgs::Int16 comp_msg;
ros::Publisher pub_comp("heading", &comp_msg);

std_msgs::Int16 depth_msg;
ros::Publisher pub_depth("depth", &depth_msg);

ros::Subscriber<std_msgs::String> sub("morseTopic", &messageCb );

typedef union accel_t_gyro_union
{
  struct
  {
    uint8_t x_accel_h;
    uint8_t x_accel_l;
    uint8_t y_accel_h;
    uint8_t y_accel_l;
    uint8_t z_accel_h;
    uint8_t z_accel_l;
    uint8_t t_h;
    uint8_t t_l;
    uint8_t x_gyro_h;
    uint8_t x_gyro_l;
    uint8_t y_gyro_h;
    uint8_t y_gyro_l;
```

```
    uint8_t z_gyro_h;
    uint8_t z_gyro_l;
  } reg;
  struct
  {
    int16_t x_accel;
    int16_t y_accel;
    int16_t z_accel;
    int16_t temperature;
    int16_t x_gyro;
    int16_t y_gyro;
    int16_t z_gyro;
  } value;
};


void setup()
{
  int error;
  uint8_t c;

  pinMode(LEDpin, OUTPUT) ;

  Serial.begin(57600);

  nh.initNode();
  nh.advertise(pub_tilt_x);
  nh.advertise(pub_tilt_y);
  nh.advertise(pub_tilt_z);
  nh.advertise(pub_comp);
  nh.advertise(pub_depth);
  nh.subscribe(sub);
```

```
  Wire.begin();

  compass.init();
  compass.enableDefault();

  compass.m_min = (LSM303::vector<int16_t>){-32767, -32767, -32767};
  compass.m_max = (LSM303::vector<int16_t>){+32767, +32767, +32767};

  error = MPU6050_read (MPU6050_WHO_AM_I, &c, 1);
  error = MPU6050_read (MPU6050_PWR_MGMT_1, &c, 1);
  MPU6050_write_reg (MPU6050_PWR_MGMT_1, 0);
}

long publisher_timer;

void messageCb( const std_msgs::String::ConstPtr& msg){
  morseMsg = msg->data.c_str()
  sendmsg(morseMsg);
}

int returnDepth(int sensorReading){
  float temp = sensorReading;
  double outInG = (1023 - (14110 * pow(temp, -0.6236)));
  int grams = outInG;
  return grams;
}

void dash()
{
  digitalWrite(LEDpin, HIGH) ;
  delay(DASHLEN);
  digitalWrite(LEDpin, LOW) ;
  delay(DOTLEN) ;
```

```
}

void dit()
{
  digitalWrite(LEDpin, HIGH) ;
  delay(DOTLEN);
  digitalWrite(LEDpin, LOW) ;
  delay(DOTLEN);
}

void send(char c)
{
  int i ;
  if (c == ' ') {
    Serial.print(c) ;
    delay(7*DOTLEN) ;
    return ;
  }
  for (i=0; i<N_MORSE; i++) {
    if (morsetab[i].c == c) {
      unsigned char p = morsetab[i].pat ;
      Serial.print(morsetab[i].c) ;

      while (p != 1) {
          if (p & 1)
            dash() ;
          else
            dit() ;
          p = p / 2 ;
      }
      delay(2*DOTLEN) ;
      return ;
    }
```

```
  }

  Serial.print("?") ;
}

void sendmsg(char *str)
{
  while (*str)
    send(*str++) ;
  Serial.println("");
}


int MPU6050_read(int start, uint8_t *buffer, int size)
{
  int i, n, error;

  Wire.beginTransmission(MPU6050_I2C_ADDRESS);
  n = Wire.write(start);
  if (n != 1)
    return (-10);

  n = Wire.endTransmission(false);
  if (n != 0)
    return (n);

  Wire.requestFrom(MPU6050_I2C_ADDRESS, size, true);
  i = 0;
  while(Wire.available() && i<size)
  {
    buffer[i++]=Wire.read();
  }
  if ( i != size)
```

```
    return (-11);

  return (0);
}


int MPU6050_write(int start, const uint8_t *pData, int size)
{
  int n, error;

  Wire.beginTransmission(MPU6050_I2C_ADDRESS);
  n = Wire.write(start);
  if (n != 1)
    return (-20);

  n = Wire.write(pData, size);
  if (n != size)
    return (-21);

  error = Wire.endTransmission(true);
  if (error != 0)
    return (error);

  return (0);
}


int MPU6050_write_reg(int reg, uint8_t data)
{
  int error;

  error = MPU6050_write(reg, &data, 1);
```

```
  return (error);
}

void loop()
{
  int error;
  double dT;
  accel_t_gyro_union accel_t_gyro;
  int sensorPin = A0;

  uint8_t swap;
  #define SWAP(x,y) swap = x; x = y; y = swap

  SWAP (accel_t_gyro.reg.x_accel_h, accel_t_gyro.reg.x_accel_l);
  SWAP (accel_t_gyro.reg.y_accel_h, accel_t_gyro.reg.y_accel_l);
  SWAP (accel_t_gyro.reg.z_accel_h, accel_t_gyro.reg.z_accel_l);
  SWAP (accel_t_gyro.reg.t_h, accel_t_gyro.reg.t_l);
  SWAP (accel_t_gyro.reg.x_gyro_h, accel_t_gyro.reg.x_gyro_l);
  SWAP (accel_t_gyro.reg.y_gyro_h, accel_t_gyro.reg.y_gyro_l);
  SWAP (accel_t_gyro.reg.z_gyro_h, accel_t_gyro.reg.z_gyro_l);


  x_tilt_msg.data = accel_t_gyro.value.x_accel;
  y_tilt_msg.data = accel_t_gyro.value.y_accel;
  z_tilt_msg.data = accel_t_gyro.value.z_accel;

  compass.read();
  int myHeading = compass.heading();
  comp_msg.data = myHeading;

  sensorReading = analogRead(sensorPin);
  depth_msg.data = returnDepth(sensorReading);
```

```
  pub_tilt_x.publish(&x_tilt_msg);
  pub_tilt_y.publish(&y_tilt_msg);
  pub_tilt_z.publish(&z_tilt_msg);
  pub_comp.publish(&comp_msg);
  pub_depth.publish(&depth_msg);
  nh.spinOnce();


  // Uncomment to print the raw gyro values.
  /*
  Serial.print(F("gyro x,y,z : "));
  Serial.print(accel_t_gyro.value.x_gyro, DEC);
  Serial.print(F(", "));
  Serial.print(accel_t_gyro.value.y_gyro, DEC);
  Serial.print(F(", "));
  Serial.print(accel_t_gyro.value.z_gyro, DEC);
  Serial.print(F(", "));
  Serial.println(F(""));
 */
  delay(100);
}
```

teleMotor.py

```
#!/usr/bin/env python
import rospy
from geometry_msgs.msg import Vector3
from sensor_msgs.msg import Joy

def callback(data):
    #define our own publisher on callback
    pub = rospy.Publisher('motor_ctl', Vector3)
    #rospy.Subscriber('joy', Joy)
    mDir = Vector3()



    #Tell us what is happening
    #rospy.loginfo("My x axis is %f" % data.axes[0])
    #rospy.loginfo("My y axis is %f" % data.axes[1])
    #rospy.loginfo("My z axis is %f" % data.axes[2])
    #ourVector = geometry_msgs.msg.Vector3

    if(data.axes[0] >= 0):
        mDir.x = data.axes[0]
        mDir.y = -1 * data.axes[0]
    else:
        mDir.x = -1 * data.axes[0]
        mDir.y = data.axes[0]

    mDir.x += data.axes[1]
    mDir.y += data.axes[1]

    if(mDir.x > 1):
        mDir.x = 1
```

```
    if(mDir.x < -1):
        mDir.x = -1


    if(mDir.y > 1):
        mDir.y = 1

    if(mDir.y < -1):
        mDir.y = -1


    mDir.z = data.axes[4]

    #Tell us what is happening
    rospy.loginfo("My left motor is %f" % (mDir.x * 100))
    rospy.loginfo("My right motor is %f" % (mDir.y * 100))
    rospy.loginfo("My vertical is %f" % (mDir.z * 100))


    pub.publish(mDir)

def listener():
    rospy.init_node('listener', anonymous=True)
    rospy.Subscriber("joy", Joy, callback)
    rospy.spin()


if __name__ == '__main__':
    listener()
```

autoMotor.py

```
#!/usr/bin/env python
import rospy
from geometry_msgs.msg import Vector3

mDir = Vector3()
mDir.x = 0
mDir.y = 0
mDir.z = 0
initialArea = 0



def callback1(data):
    #define our own publisher on callback
    pub = rospy.Publisher('motor_ctl', Vector3)

    if((data.x -160) >= 0):
        mDir.x = (data.x - 160)
        mDir.y = -1 * (data.x - 160)
    else:
        mDir.x = -1 * (data.x -160)
        mDir.y = (data.x - 160)

    mDir.x += (data.y - 120)
    mDir.y += (data.y - 120)

    mDir.x = (mDir.x)/280
    mDir.y = (mDir.y)/280

    if(mDir.x > 1):
        mDir.x = 1
```

```
    if(mDir.x < -1):
        mDir.x = -1

    if(mDir.y > 1):
        mDir.y = 1

    if(mDir.y < -1):
        mDir.y = -1

    if(data.y == 0):
        data.y = 1

    if(data.y >= 240):
       mDir.z = -1 * (abs(data.y-240)/240)
    else:
        mDir.z = 1 * (abs(data.y-240)/240)



    #Tell us what is happening
    rospy.loginfo("My left motor is %f" % (mDir.x * 100))
    rospy.loginfo("My right motor is %f" % (mDir.y * 100))
    rospy.loginfo("My vertical is %f" % (mDir.z * 100))

    pub.publish(mDir)

def callback2(data):
    #define our own publisher on callback
    pub = rospy.Publisher('motor_ctl', Vector3)
    if(initialArea == 0):
        initialArea = data.x * data.y
    currentArea = data.x * data.y
    thrust = (initialArea - currentArea)/initialArea
```

```
    if(thrust > 1):
        thrust = 1
    if(thrust < -1):
        thrust = -1

    mDir.x = thrust
    mDir.y = -1 * thrust

    if(mDir.x > 1):
        mDir.x = 1

    if(mDir.x < -1):
        mDir.x = -1

    if(mDir.y > 1):
        mDir.y = 1

    if(mDir.y < -1):
        mDir.y = -1

    #Tell us what is happening
    rospy.loginfo("My left motor is %f" % (mDir.x * 100))
    rospy.loginfo("My right motor is %f" % (mDir.y * 100))

    pub.publish(mDir)


def listener1():
    rospy.init_node('listener1', anonymous=True)
    rospy.Subscriber("my_vector", Vector3, callback1)
    rospy.spin()

def listener2():
```

```
    rospy.init_node('listener2', anonymous=True)
    rospy.Subscriber("my_vector_area", Vector3, callback2)
    rospy.spin()


if __name__ == '__main__':
    listener1()
    listener2()
```

The following code is modified from the ROS OpenTLD release which can be found here:

https://github.com/Ronan0912/ros_opentld

In order to use, simple replace the files found in the source with the code below

base_frame.hpp

```
/*  Copyright 2012 UdeS University of Sherbrooke
 *
 *   This file is part of ROS_OpenTLD.
 *
 *   ROS_OpenTLD is free software: you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *    the Free Software Foundation, either version 3 of the License, or
 *   (at your option) any later version.
 *
 *   ROS_OpenTLD is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *   GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with ROS_OpenTLD. If not, see <http://www.gnu.org/licenses/>.
 *
 */
/*
 * base_frame.hpp
 *
 *  Created on: May 17, 2012
 *      Author: Ronan Chauvin
 *
 *  Modified by Christopher Conley and Jillian Chalke
 */
```

```
#ifndef BASE_FRAME_H_
#define BASE_FRAME_H_

#include <QtGui/QWidget>
#include <QtGui/QKeyEvent>
#include <QtCore/QObject>
#include <QtGui/QImage>
#include <QtCore/QRectF>

#include <ros/ros.h>
#include <cv_bridge/cv_bridge.h>
#include <sensor_msgs/Image.h>
#include <tld_msgs/Target.h>
#include <tld_msgs/BoundingBox.h>
#include <std_msgs/Char.h>
#include <std_msgs/Float32.h>

#include "ui_baseFrame.h"

class BaseFrame : public QWidget, private Ui::BaseFrame
{
	Q_OBJECT

	public:
		ros::NodeHandle n;
		ros::Publisher pub1;
		ros::Publisher pub2;
		ros::Publisher pub3;
		ros::Publisher pub4;
		ros::Subscriber sub1;
		ros::Subscriber sub2;
		ros::Subscriber sub3;
```

```
        BaseFrame();
        virtual ~BaseFrame();

    protected:
        void keyPressEvent(QKeyEvent * event);

    private:
        cv_bridge::CvImageConstPtr cv_ptr;
        bool first_image;

        void image_receivedCB(const sensor_msgs::ImageConstPtr & msg);
        void tracked_objectCB(const tld_msgs::BoundingBoxConstPtr & msg);
        void fps_trackerCB(const std_msgs::Float32ConstPtr & msg);

    signals:
        void sig_image_received(const QImage & image);
        void sig_tracked_object_changed(const QRectF & bb);
        void sig_fps_tracker_changed(int fps);
        void sig_confidence_changed(int confidence);

        public slots:
        void clear_background();
        void clear_and_stop_tracking();
        void toggle_learning();
        void alternating_mode();
        void export_model();
        void import_model();
        void reset();
};

#endif
```

base_frame.cpp

```
/*  Copyright 2012 UdeS University of Sherbrooke
 *
 *   This file is part of ROS_OpenTLD.
 *
 *   ROS_OpenTLD is free software: you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *    the Free Software Foundation, either version 3 of the License, or
 *   (at your option) any later version.
 *
 *   ROS_OpenTLD is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *   GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with ROS_OpenTLD. If not, see <http://www.gnu.org/licenses/>.
 *
 */
/*
 * base_frame.cpp
 *
 *  Created on: May 17, 2012
 *      Author: Ronan Chauvin
 *
 *    Modified by Christopher Conley and Jillian Chalke
 */

#include "base_frame.hpp"
#include <sensor_msgs/image_encodings.h>
#include <geometry_msgs/Vector3.h>
```

```
#include <QtCore/QDebug>

#include <QtCore/QString>

#include <QtGui/QFileDialog>


namespace enc = sensor_msgs::image_encodings;


BaseFrame::BaseFrame()

{

	setupUi(this);


	QObject::connect(this,SIGNAL(sig_image_received(const QImage
&)),base_frame_graphics_view,SLOT(image_received(const QImage &)));

	QObject::connect(this,SIGNAL(sig_tracked_object_changed(const QRectF
&)),base_frame_graphics_view,SLOT(tracked_objet_changed(const QRectF &)));

	QObject::connect(this,SIGNAL(sig_fps_tracker_changed(int)),lcd_fps_tracker,SLO
T(display(int)));

	QObject::connect(this,SIGNAL(sig_confidence_changed(int)),confidence_bar,SLOT(
setValue(int)));

	QObject::connect(background_reset_button,SIGNAL(clicked()),this,SLOT(clear_bac
kground()));

	QObject::connect(learning_button,SIGNAL(clicked()),this,SLOT(toggle_learning()
));

	QObject::connect(alternating_button,SIGNAL(clicked()),this,SLOT(alternating_mo
de()));

	QObject::connect(stop_tracking_button,SIGNAL(clicked()),this,SLOT(clear_and_st
op_tracking()));

	QObject::connect(importing_button,SIGNAL(clicked()),this,SLOT(import_model()))
;

	QObject::connect(exporting_button,SIGNAL(clicked()),this,SLOT(export_model()))
;

	QObject::connect(reset_button,SIGNAL(clicked()),this,SLOT(reset()));


	sub1 = n.subscribe("image", 1000, &BaseFrame::image_receivedCB, this);

	sub2 = n.subscribe("tracked_object", 1000, &BaseFrame::tracked_objectCB,
this);

	sub3 = n.subscribe("fps_tracker", 1000, &BaseFrame::fps_trackerCB, this);

	pub1 = n.advertise<tld_msgs::Target>("tld_gui_bb", 1000, true);
```

```
	pub2 = n.advertise<std_msgs::Char>("tld_gui_cmds", 1000, true);
	 pub3 = n.advertise<geometry_msgs::Vector3>("my_vector", 1000, true);
	 pub4 = n.advertise<geometry_msgs::Vector3>("my_vector_area", 1000, true);

	first_image = true;
}

BaseFrame::~BaseFrame()
{

}

void BaseFrame::keyPressEvent(QKeyEvent * event)
{
	switch (event->key())
	{
		case Qt::Key_Return:
		case Qt::Key_Enter:
		case Qt::Key_Backspace:
			qDebug() << "Enter";
			if(!base_frame_graphics_view->get_bb()->rect().isEmpty())
			{
				tld_msgs::Target msg;
				msg.bb.x = (int)base_frame_graphics_view->get_bb()-
>rect().x();
				msg.bb.y = (int)base_frame_graphics_view->get_bb()-
>rect().y();
				msg.bb.width = (int)base_frame_graphics_view->get_bb()-
>rect().width();
				msg.bb.height = (int)base_frame_graphics_view->get_bb()-
>rect().height();
				msg.bb.confidence = 1.0;
				cv_ptr->toImageMsg(msg.img);
				pub1.publish(msg);
```

```
            }
            break;
        case Qt::Key_Q:
            qDebug() << "Quitting";
            close();
            break;
        case Qt::Key_B:
            clear_background();
            break;
        case Qt::Key_C:
            clear_and_stop_tracking();
            break;
        case Qt::Key_L:
            toggle_learning();
            break;
        case Qt::Key_A:
            alternating_mode();
            break;
        case Qt::Key_E:
            export_model();
            break;
        case Qt::Key_I:
            import_model();
            break;
        case Qt::Key_R:
            reset();
            break;
        case Qt::Key_F5:
            first_image = true;
        default:
            event->ignore();
            break;
}
```

```
}

void BaseFrame::image_receivedCB(const sensor_msgs::ImageConstPtr & msg)
{
	if(first_image || base_frame_graphics_view->get_correct_bb())
	{
		try
		{
			if (enc::isColor(msg->encoding))
				cv_ptr = cv_bridge::toCvShare(msg, enc::RGB8);
			else
				cv_ptr = cv_bridge::toCvShare(msg, enc::MONO8);
		}
		catch (cv_bridge::Exception& e)
		{
			ROS_ERROR("cv_bridge exception: %s", e.what());
			return;
		}

		QImage image;

		if (enc::isColor(msg->encoding))
		{
			image = QImage((const unsigned char*)(cv_ptr->image.data),cv_ptr-
>image.cols,cv_ptr->image.rows,QImage::Format_RGB888);
			//image.rgbSwapped();
		}
		else
		{
			image = QImage((const unsigned char*)(cv_ptr->image.data),cv_ptr-
>image.cols,cv_ptr->image.rows,QImage::Format_Indexed8);
		}

		emit sig_image_received(image);
```

```
            if(first_image)
                  first_image = false;
      }
}

void BaseFrame::tracked_objectCB(const tld_msgs::BoundingBoxConstPtr & msg)
{
    if(msg->width && msg->height)
        first_image = true;
    else
        first_image = false;

      QRectF rect(msg->x,msg->y,msg->width,msg->height);
      emit sig_tracked_object_changed(rect);
      emit sig_confidence_changed((int)(msg->confidence*100));
}

void BaseFrame::fps_trackerCB(const std_msgs::Float32ConstPtr & msg)
{
      emit sig_fps_tracker_changed((int)msg->data);
}

void BaseFrame::clear_background()
{
      std_msgs::Char cmd;
      cmd.data = 'b';
      pub2.publish(cmd);
      qDebug() << "Clearing Background";
}

void BaseFrame::clear_and_stop_tracking()
{
```

```
	std_msgs::Char cmd;
	cmd.data = 'c';
	pub2.publish(cmd);
	qDebug() << "Clearing and stop tracking";
}

void BaseFrame::toggle_learning()
{
	std_msgs::Char cmd;
	cmd.data = 'l';
	pub2.publish(cmd);
	qDebug() << "Toggle learning";
}

void BaseFrame::alternating_mode()
{
	std_msgs::Char cmd;
	cmd.data = 'a';
	pub2.publish(cmd);
	qDebug() << "Alternating mode";
}

void BaseFrame::export_model()
{
	QString res = QFileDialog::getSaveFileName(this, tr("Choose a model file
name"), "/", tr("All (*)"));
	if(!res.isNull())
	{
		ros::NodeHandle nh;
		nh.setParam("/ros_tld_tracker_node/modelExportFile", res.toStdString());
	}
	else
	{
```

```
		// The user pressed cancel or closed the dialog.
	}


	std_msgs::Char cmd;
	cmd.data = 'e';
	pub2.publish(cmd);
	qDebug() << "Exporting model : " << res;
}


void BaseFrame::import_model()
{
	QString res = QFileDialog::getOpenFileName(this, tr("Choose a model file to
open"), "/", tr("All (*)"));
	if(!res.isNull())
	{
		ros::NodeHandle nh;
		nh.setParam("/ros_tld_tracker_node/modelImportFile", res.toStdString());
	}
	else
	{
		// The user pressed cancel or closed the dialog.
	}


	std_msgs::Char cmd;
	cmd.data = 'i';
	pub2.publish(cmd);
	qDebug() << "Importing model : " << res;
}


void BaseFrame::reset()
{
	std_msgs::Char cmd;
	cmd.data = 'r';
```

```
	pub2.publish(cmd);
	qDebug() << "Reset";
}
```

main.hpp

```
/*  Copyright 2012 UdeS University of Sherbrooke
 *
 *   This file is part of ROS_OpenTLD.
 *
 *   ROS_OpenTLD is free software: you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *    the Free Software Foundation, either version 3 of the License, or
 *   (at your option) any later version.
 *
 *   ROS_OpenTLD is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *   GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with ROS_OpenTLD. If not, see <http://www.gnu.org/licenses/>.
 *
 */
/*
 * main.hpp
 *
 *  Created on: June 8, 2012
 *      Author: Ronan Chauvin
 *
 *    Modified by Christopher Conley and Jillian Chalke
 */
```

```
#ifndef MAIN_HPP_
#define MAIN_HPP_

#include <tld/TLD.h>
#include <boost/interprocess/sync/interprocess_mutex.hpp>

#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/objdetect/objdetect.hpp>

#include <ros/ros.h>
#include <cv_bridge/cv_bridge.h>
#include <std_msgs/Header.h>
#include <sensor_msgs/Image.h>
#include <tld_msgs/Target.h>
#include <tld_msgs/BoundingBox.h>
#include <std_msgs/Char.h>
#include <std_msgs/Float32.h>
#include <geometry_msgs/Vector3.h>
#include <string>

class Main
{
	public:
		Main()
		{
			tld = new tld::TLD();
			state = INIT;

			ros::NodeHandle np("~");
			np.param("showOutput", showOutput, true);
			np.param("loadModel", loadModel, false);
```

```
                np.param("autoFaceDetection", autoFaceDetection, false);
                np.param("exportModelAfterRun", exportModelAfterRun, false);
                np.param("modelImportFile", modelImportFile,
std::string("model"));
                np.param("modelExportFile", modelExportFile,
std::string("model"));
                np.param("cascadePath", face_cascade_path,
                        std::string("haarcascade_frontalface_alt.xml"));

                np.param("x", target_bb.x, 100);
                np.param("y", target_bb.y, 100);
                np.param("width", target_bb.width, 100);
                np.param("height", target_bb.height, 100);
                np.param("correctBB", correctBB, false);

                pub1 = n.advertise<tld_msgs::BoundingBox>("tld_tracked_object",
1000, true);
                pub2 = n.advertise<std_msgs::Float32>("tld_fps", 1000, true);
                pub3 = n.advertise<geometry_msgs::Vector3>("my_vector", 1000,
true);
                pub4 = n.advertise<geometry_msgs::Vector3>("my_vector_area",
1000, true);
                sub1 = n.subscribe("image", 1000, &Main::imageReceivedCB, this);
                sub2 = n.subscribe("bounding_box", 1000, &Main::targetReceivedCB,
this);
                sub3 = n.subscribe("cmds", 1000, &Main::cmdReceivedCB, this);

                semaphore.lock();
        }

        ~Main()
        {
                delete tld;
        }
```

```
        void process();

private:
        tld::TLD * tld;
        bool showOutput;
        bool exportModelAfterRun;
        bool loadModel;
        bool autoFaceDetection;
        std::string modelImportFile;
        std::string modelExportFile;

        enum
        {
                INIT,
                TRACKER_INIT,
                TRACKING,
                STOPPED
        } state;

        bool correctBB;
        cv::Rect target_bb;
        cv::Mat target_image;

        std_msgs::Header img_header;
        cv::Mat img;
        cv_bridge::CvImagePtr img_buffer_ptr;
        cv::Mat gray;
        boost::interprocess::interprocess_mutex mutex;
        boost::interprocess::interprocess_mutex semaphore;
        ros::NodeHandle n;
        ros::Publisher pub1;
        ros::Publisher pub2;
        ros::Publisher pub3;
```

```
            ros::Publisher pub4;
            ros::Subscriber sub1;
            ros::Subscriber sub2;
            ros::Subscriber sub3;

            std::string face_cascade_path;
            cv::CascadeClassifier face_cascade;

            bool newImageReceived();
            void getLastImageFromBuffer();
            void imageReceivedCB(const sensor_msgs::ImageConstPtr & msg);
            void targetReceivedCB(const tld_msgs::TargetConstPtr & msg);
            void cmdReceivedCB(const std_msgs::CharConstPtr & cmd);
            void sendObjectTracked(int x, int y, int width, int height, float
confidence);

            void clearBackground();
            void stopTracking();
            void toggleLearning();
            void alternatingMode();
            void exportModel();
            void importModel();
            void reset();

            cv::Rect faceDetection();
};

#endif /* MAIN_HPP_ */
```

main.cpp

```
/*  Copyright 2012 UdeS University of Sherbrooke
 *
```

```
 *   This file is part of ROS_OpenTLD.
 *
 *   ROS_OpenTLD is free software: you can redistribute it and/or modify
 *   it under the terms of the GNU General Public License as published by
 *    the Free Software Foundation, either version 3 of the License, or
 *   (at your option) any later version.
 *
 *   ROS_OpenTLD is distributed in the hope that it will be useful,
 *   but WITHOUT ANY WARRANTY; without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *   GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *   along with ROS_OpenTLD. If not, see <http://www.gnu.org/licenses/>.
 *
 */
/*
 * main.cpp
 *
 *  Created on: June 8, 2012
 *      Author: Ronan Chauvin
 *
 *    Modified by Christopher Conley and Jillian Chalke
 */


#include "main.hpp"
#include <ros/time.h>
#include <ros/duration.h>
#include <sensor_msgs/image_encodings.h>
#include <geometry_msgs/Vector3.h>
```

```
namespace enc = sensor_msgs::image_encodings;

void Main::process()
{
	if(autoFaceDetection && !face_cascade.load(face_cascade_path))
	{
		ROS_FATAL("--(!)Error loading cascade detector\n");
		return;
	}

	while (ros::ok())
	{
		switch (state)
		{
			case INIT:
				if(newImageReceived())
				{
					if(showOutput)
						sendObjectTracked(0, 0, 0, 0, 0);
					getLastImageFromBuffer();
					tld->detectorCascade->imgWidth = gray.cols;
					tld->detectorCascade->imgHeight = gray.rows;
					tld->detectorCascade->imgWidthStep = gray.step;

					state = TRACKER_INIT;
				}
				break;
			case TRACKER_INIT:
				if(loadModel && !modelImportFile.empty())
				{
					ROS_INFO("Loading model %s",
modelImportFile.c_str());
```

```
                                tld->readFromFile(modelImportFile.c_str());
                                tld->learningEnabled = false;
                                state = TRACKING;
                        }
                        else if(autoFaceDetection || correctBB)
                        {
                                if(autoFaceDetection)
                                {
                                        target_image = gray;
                                        target_bb = faceDetection();
                                }

        sendObjectTracked(target_bb.x,target_bb.y,target_bb.width,target_bb.height,1.0
);

                                ROS_INFO("Starting at %d %d %d %d\n", target_bb.x,
target_bb.y, target_bb.width, target_bb.height);

                                tld->selectObject(target_image, &target_bb);
                                tld->learningEnabled = true;
                                state = TRACKING;
                        }
                        else
                        {
                                ros::Duration(1.0).sleep();
                                ROS_INFO("Waiting for a BB");
                        }
                        break;
                case TRACKING:
                        if(newImageReceived())
                        {
                                ros::Time tic = ros::Time::now();
```

```
                                getLastImageFromBuffer();
                                tld->processImage(img);

                                ros::Duration toc = (ros::Time::now() - tic);
                                float fps = 1.0/toc.toSec();

                                std_msgs::Float32 msg_fps;
                                msg_fps.data = fps;
                                pub2.publish(msg_fps);

                                if(showOutput && tld->currBB != NULL)
                                {
                                        sendObjectTracked(tld->currBB->x,tld->currBB->y,tld->currBB->width,tld->currBB->height,tld->currConf);
                                }
                        }
                        break;
                case STOPPED:
                                ros::Duration(1.0).sleep();
                                ROS_INFO("Tracker stopped");
                        break;
                default:
                        break;
                }
        }

        if(exportModelAfterRun)
        {
                tld->writeToFile(modelExportFile.c_str());
        }

        semaphore.unlock();
}
```

```
void Main::imageReceivedCB(const sensor_msgs::ImageConstPtr & msg)
{
	bool empty = false;
	mutex.lock();

	if(img_buffer_ptr.get() == 0)
	{
		empty = true;
	}

	try
	{
		if (enc::isColor(msg->encoding))
			img_buffer_ptr = cv_bridge::toCvCopy(msg, enc::RGB8);
		else
		{
			img_buffer_ptr = cv_bridge::toCvCopy(msg, enc::MONO8);
			cv::cvtColor(img_buffer_ptr->image, img_buffer_ptr->image,
CV_GRAY2BGR);
		}
	}
	catch (cv_bridge::Exception& e)
	{
		ROS_ERROR("cv_bridge exception: %s", e.what());
		return;
	}

	if(empty)
	{
		semaphore.unlock();
	}
	mutex.unlock();
```

```
}

void Main::targetReceivedCB(const tld_msgs::TargetConstPtr & msg)
{
	reset();
	ROS_ASSERT(msg->bb.x >= 0);
	ROS_ASSERT(msg->bb.y >= 0);
	ROS_ASSERT(msg->bb.width > 0);
	ROS_ASSERT(msg->bb.height > 0);
	ROS_INFO("Bounding Box received");

	target_bb.x = msg->bb.x;
	target_bb.y = msg->bb.y;
	target_bb.width = msg->bb.width;
	target_bb.height = msg->bb.height;

	try
	{
		target_image = cv_bridge::toCvCopy(msg->img, enc::MONO8)->image;
	}
	catch (cv_bridge::Exception& e)
	{
		ROS_ERROR("cv_bridge exception: %s", e.what());
		return;
	}

	correctBB = true;
}

void Main::cmdReceivedCB(const std_msgs::CharConstPtr & cmd)
{
	switch (cmd->data)
	{
```

```
		case 'b':
			clearBackground();
			break;
		case 'c':
			stopTracking();
			break;
		case 'l':
			toggleLearning();
			break;
		case 'a':
			alternatingMode();
			break;
		case 'e':
			exportModel();
			break;
		case 'i':
			importModel();
			break;
		case 'r':
			reset();
			break;
		default:
			break;
	}
}

void Main::sendObjectTracked(int x, int y, int width, int height, float confidence)
{
	tld_msgs::BoundingBox msg;
	msg.header = img_header; //Add the Header of the last image processed
	msg.x = x;
	msg.y = y;
	msg.width = width;
```

```
        msg.height = height;
        msg.confidence = confidence;
        pub1.publish(msg);
          geometry_msgs::Vector3 mymsg;
          geometry_msgs::Vector3 mymsgarea;
          mymsg.x = x;
          mymsg.y = y;
        mymsgarea.x = width;
        mymsgarea.x = height;
          pub3.publish(mymsg);
          pub4.publish(mymsgarea);
}

bool Main::newImageReceived()
{
        semaphore.lock();
        return true;
}

void Main::getLastImageFromBuffer()
{
        mutex.lock();
        img_header = img_buffer_ptr->header;
        img = img_buffer_ptr->image;

        cv::cvtColor(img, gray, CV_BGR2GRAY);

        img_buffer_ptr.reset();
        mutex.unlock();
}

void Main::clearBackground()
{
```

```
	tld::ForegroundDetector* fg = tld->detectorCascade->foregroundDetector;

	if(fg->bgImg.empty())
	{
		gray.copyTo(fg->bgImg);
	}
	else
	{
		fg->bgImg.release();
	}
}

void Main::stopTracking()
{
	if(state == STOPPED)
	  state = TRACKING;
    else
	  state = STOPPED;
}

void Main::toggleLearning()
{
	tld->learningEnabled = !tld->learningEnabled;
	ROS_INFO("LearningEnabled: %d\n", tld->learningEnabled);
}

void Main::alternatingMode()
{
	tld->alternating = !tld->alternating;
	ROS_INFO("Alternating: %d\n", tld->alternating);
}

void Main::exportModel()
```

```
{
	ros::NodeHandle np("~");
	np.getParam("modelExportFile", modelExportFile);
	//tld->learningEnabled = false;
	tld->writeToFile(modelExportFile.c_str());
	ROS_INFO("Exporting model %s", modelExportFile.c_str());
}

void Main::importModel()
{
	ros::NodeHandle np("~");
	np.getParam("modelImportFile", modelImportFile);
	loadModel = true;
	state = TRACKER_INIT;
}

void Main::reset()
{
	correctBB = false;
	state = INIT;
}

cv::Rect Main::faceDetection()
{
	std::vector<cv::Rect> faces;

	while(faces.empty())
	{
		if(newImageReceived())
			getLastImageFromBuffer();

		cv::equalizeHist(gray, gray);
```

```
            face_cascade.detectMultiScale(gray, faces, 1.1, 2,
0|CV_HAAR_SCALE_IMAGE, cv::Size(30, 30));
      }


      return faces[0];
}
```